FARMACOLOGIA
FÍSICA
GENÈTICA
DRET
GEOLOGIA
INFORMÀTICA
FILOLOGIA
VETERINÀRIA
PEDIATRIA
PERIODISME
BIOLOGIA
SOCIOLOGIA
BOTÀNICA
HISTÒRIA
ECONOMIA
MATEMÀTIQUES
MEDICINA
ART
FILOSOFIA
GEOGRAFIA
PEDAGOGIA
QUÍMICA
ELECTRÒNICA
MÚSICA
BIOTECNOLOGIA
ANTROPOLOGIA
ZOOLOGIA
PSICOLOGIA
PSIQUIATRIA

Universitat Autònoma de Barcelona
Servei de Publicacions
Departament de Microelectrònica
i Sistemes Electrònics

Joaquim Saiz
Antoni Portero
Raül Aragonès

Desenvolupament del microprocessador *LittleProc*: des de la primera porta lògica fins a una plataforma reconfigurable

col·lecció MATERIALS 234

Joaquim Saiz, Antoni Portero, Raül Aragonès

Desenvolupant el microprocessador *LittleProc*: des de la primera porta lògica fins a una plataforma reconfigurable

Universitat Autònoma de Barcelona
Servei de Publicacions
Bellaterra, 2015

Primera edició: octubre de 2015
Edició i impressió:
Servei de Publicacions
Universitat Autònoma de Barcelona
Edifici A. 08193 Bellaterra (Barcelona). Spain
sp@uab.cat
http://publicacions.uab.cat/

Imprès a Espanya. Printed in Spain

Dipòsit legal (versió electrònica): B. 25363-2015
ISBN (versió electrònica): 978-84-490-2699-7

Índex

0. Presentació del llibre 9
   0.1 Els orígens 9
   0.2 Assignatura i estudiants *target* 10
   0.3 Pràctiques: objectius i metodologia global 10
   0.4 Estructuració de les sessions 11
   0.5 Estructura del llibre 12

1. Primeres passes 15
   1.1 Ajuda en línia de Quartus 16
   1.2 Creació d'un nou projecte 17

2. Introducció del disseny mitjançant captura d'esquemes 23
   2.1 Invocació de l'editor d'esquemes de Quartus 23
   2.2 Addició de portes lògiques 25
   2.3 Operació amb símbols (selecció, moviment i rotació) 26
   2.4 Addició de terminals (*ports*) d'entrada i sortida 26
   2.5 Addició de línies de connexió 27
   2.6 Gravació de l'esquema 28
   2.7 Creació d'un símbol i modificació de l'ordre dels seus pins 29
   2.8 Tancament del projecte 30
   2.9 Obertura d'un projecte que ja està creat 30

3. Compilació 31

4. Simulació 33
   4.1 Introducció 33
   4.2 Simulació funcional 34
      4.2.1 Configuració inicial i generació de la *netlist* 34
      4.2.2 Introducció dels vectors d'entrada 35
      4.2.3 Realització de la simulació i visualització de resultats 39
   4.3 Simulació amb retards (*timing simulation*) 40
      4.3.1 Configuració inicial 40
      4.3.2 Elecció dels senyals 42
      4.3.3 Especificació d'una forma d'ona 44
         4.3.3.1 Aplicació d'una sèrie de valors numèrics successius a un grup de senyals o bus 44
         4.3.3.2 Aplicació d'un valor constant a una entrada 46
         4.3.3.3 Aplicació d'una forma d'ona periòdica 46
         4.3.3.4 Especificació d'una forma d'ona qualsevol 46

4.3.4 Gravació del fitxer d'ones. Especificació del fitxer d'entrada de simulació ... 47
4.3.5 Realització de la simulació ... 48
4.4 Simulacions addicionals ... 49

5. Alguns mòduls bàsics de *LittleProc* ... 51
5.1 Introducció ... 51
5.2 Nom de cel·la i nom d'instància ... 51
5.3 Baldes (*Latches*) ... 51
5.3.1 Funcionament d'una balda (o *latch*) ... 51
5.4 Registre de baldes (o *latches*) ... 52
5.5 *Flip-flops* ... 53
5.5.1 Funcionament d'un *flip-flop* ... 53
5.6 Connexió per nom ... 54
5.7 Especificació dels busos ... 56
5.8 Expressió compacta d'un conjunt d'instàncies ... 56
5.9 Cel·les a realitzar ... 57
5.9.1 Registre de baldes de 8 bits ... 57
5.9.2 Registre de *flip-flops* de 8 bits (cel·la microPC) ... 57
5.9.3 Cel·la PC ... 58

6. VHDL llenguatge de descripció de maquinari ... 59
6.1 Introducció ... 59
6.2 El llenguatge VHDL ... 61
6.2.1 Conceptes bàsics ... 61
6.2.2 Utilització del llenguatge de descripció de maquinari ... 62
6.2.3 Definició de mòduls en VHDL ... 62
6.2.4 Unitats de disseny en VHDL ... 63
6.2.4.1 Entitat ... 63
6.2.4.2 Arquitectura ... 64
6.2.4.3 Configuracions ... 65
6.2.4.4 Paquets ... 66
6.2.5 Biblioteques ... 66
6.2.6 Variables i senyals ... 67
6.2.6.1. Variables ... 67
6.2.6.2. Senyals ... 67
6.2.6.3. Tipus de dades. Operadors ... 68
6.2.7 Atributs ... 69
6.3 Circuits combinacionals ... 70
6.3.1 Declaració d'un sumador complet d'un bit ... 70
6.3.2 Declaració d'un sumador de 8 bits a nivell comportamental ... 71
6.3.3 Multiplexor de 2 entrades: utilització de la sentència *if* ... 72
6.3.4 Taules de conversió: utilització de la sentència *case* ... 73
6.4 Mòduls d'emmagatzemament d'informació ... 75
6.4.1 Descripció d'un *latch* ... 75
6.4.2 Descripció d'un *flip-flop* ... 76
6.5 Sistemes seqüencials ... 76
6.5.1 Comptador ... 76
6.5.2 Màquines d'estats finits en VHDL ... 77
6.5.2.1. Descripció d'una màquina d'estats amb 1 procés (ús exclusiu de l'estat actual) ... 78

6.5.2.2. Descripció d'una màquina d'estats amb dos processos (ús d'estat actual i estat següent) 79
6.5.2.3. Màquines d'estats finits de Moore 80
6.5.2.4. Màquines d'estats finits de Mealy 83
6.6 Blocs elementals de LittleProc en VHDL 85
6.6.1 Realització d'un incrementador 85
6.6.2 Realització d'un registre 85
6.6.3 Realització de l'ALU (Unitat Aritmeticològica) 86
6.6.4 Implementació del bloc IRdecoder 90

7. Arquitectura interna del processador 91
7.1 Introducció 91
7.2 Característiques genèriques del processador 91
7.3 Interfície del processador 91
7.4 Arquitectura interna del processador 92
7.5 Disseny de la unitat de procés de dos busos 92
7.5.1 Descripció de la funcionalitat de cada element 93
7.5.1.1. Bus 1 93
7.5.1.2. Bus 2 94
7.5.1.3. Registres 94
7.5.1.4. ALU 99
7.5.2 Arquitectura genèrica de la UP 99
7.6 Simulació de la UP 101

8. Disseny de la unitat de control 107
8.1 Procés a dur a terme per a l'execució d'una instrucció 107
8.2 Elements bàsics de la unitat de control 108
8.3 Descripció de la funcionalitat de cada element 108
8.3.1 Seqüenciador (sequencer) 108
8.3.1.1 Multiplexor 110
8.3.1.2 microPC 110
8.3.1.3 Incrementador (inc) 111
8.3.2 Descodificador d'instruccions (IRdecoder) 111
8.3.3 Multiplexor de condicions d'entrada 112
8.3.4 ctrlSeq 113
8.3.5 Bateria de portes AND 114
8.3.6 ROM de microcodi 115
8.4 Arquitectura genèrica de la UC 117
8.5 Simulació de la UC 118

9. Conjunt d'instruccions 123

10. Disseny de la ROM de microcodi 125
10.1 Estructura de la memòria de control 125
10.2 Seccions del microcodi de control 125
10.3 Cicle d'inicialització 125
10.4 Cicle de *fetch* o de cerca de la instrucció 125
10.5 Cicles d'execució de les diverses instruccions 126
10.6 Taula de descodificació 129
10.7 Contingut de la ROM 129

11. Disseny de la RAM 131
11.1 Estructura de la memòria RAM 131
11.2 Creació de la memòria RAM 131
11.3 Estructura de les dades 132
11.4 Funcionament de la RAM 133

12. Disseny intern del processador 135
12.1 Simulació del processador 137

13. Disseny del microprocessador amb un bus 139
13.1 Arquitectura genèrica de la UP 139
13.2 Unitat de control. Microcodi 140

14. Implementació a la placa Altera UP3 Education Kit 143
14.1 Placa de prototipatge 144
14.2 Modificació de LittleProc 145
14.3 Associació de pins 149
14.4 Programació 150

15. Implementació a la placa Altera DE2 153
15.1 Placa de desenvolupament Altera DE2 154
15.2 Implementació a placa 156
15.3 Modificacions de LittleProc 157
15.3.1 Generació d'un rellotge "lent" 158
15.3.2 Accés al contingut de tots els registres 158
15.3.3 Ús de l'LCD 160
15.3.4 Visualització del rellotge del processador 161
15.4 Associació de pins 161
15.5 Programació 162

16. Col·lecció de problemes 165
16.1 Qüestionari escrit – Capítols 1 - 5 165
16.2 Qüestionari escrit – Capítol 6 166
16.3 Qüestionari escrit – Capítols 7 - 13 167
16.4 Qüestionari escrit – Capítols 14 - 15 172
16.5 Preguntes per a qüestionaris orals – Capítols 1 - 6 173
16.6 Preguntes per a qüestionaris orals – Capítols 7 - 13 174
16.7 Preguntes per a qüestionaris orals – Capítols 14 - 15 175

17. Referències i bibliografia 177
17.1 Referències relacionades amb els capítols 0 - 5 177
17.2 Referències VHDL 177
17.3 Referències relacionades amb la implementació a placa 177
17.4 Publicacions en congressos i revistes 178

Llista de figures i taules 179

0. Presentació del llibre

Aquest és un llibre que ha estat escrit per un grup de professors que imparteixen la seva docència en l'àmbit de les enginyeries informàtica i electrònica. Per tant, no ha de sorprendre gaire que haguem decidit començar la numeració dels capítols pel número 0. Per a un enginyer informàtic/electrònic el zero és el primer número, l'origen simbòlic on comencen molts processos. Aquest capítol zero està dedicat a explicar quins són els orígens d'aquest llibre, què va motivar-ne l'escriptura, quins són els seus objectius i a quines persones va dirigit. Per tant, resulta fonamental per comprendre la resta del llibre tant a escala d'estructura com de continguts. Per això recomanem la seva lectura detallada especialment a aquells professors que el pensin utilitzar per a les seves tasques docents.

0.1. Els orígens

Una màquina algorísmica consisteix en un sistema digital capaç de materialitzar i executar un algorisme determinat. La plena comprensió de les màquines algorísmiques comporta un alt nivell de dificultat per als estudiants de diversos graus d'Enginyeria (Informàtica o Electrònica). En el nostre entorn docent, durant diversos anys s'havia observat que una part important dels estudiants acabava l'assignatura on s'impartia aquesta matèria amb un coneixement no òptim de les màquines algorísmiques més complexes (les màquines microprogramades). A causa d'aquest fet, un conjunt de professors vam prendre la decisió, aprofitant la reestructuració de l'assignatura derivada de la implantació del procés de Bolonya, de definir unes noves pràctiques amb la finalitat de facilitar l'aprenentatge d'aquestes màquines. En concret, vam decidir que els estudiants implementessin de manera real una màquina algorísmica que tingués una unitat de control microprogramada. Vam pensar que la millor opció era que realitzessin un petit processador, com a exemple de màquina algorísmica de propòsit general.

En la literatura docent hi ha múltiples exemples de processadors didàctics. No obstant això, per diverses raons, vam preferir optar per un que s'adaptés als nostres interessos docents i al perfil dels nostres estudiants. Això ens va portar a desenvolupar un processador simple d'arquitectura Von Newmann clàssica, el qual vam anomenar LittleProc, que s'ajustés totalment als requeriments de la nostra assignatura i que alhora pogués servir de complement als coneixements exposats en altres assignatures relacionades amb aquest tema. La metodologia d'ensenyament pràctic, ideada i estructurada al voltant de LittleProc, és explicada més endavant en aquest capítol. Un aspecte clau de la nostra metodologia és que els nostres estudiants comptin amb una documentació adient que els permeti afrontar setmana rere setmana el repte de fer un nou pas en el disseny del processador i que els serveixi com a guia de treball. Justament aquest és

l'objectiu d'aquest llibre: constituir la documentació que servirà de guia a l'estudiant durant la realització de les pràctiques al llarg d'un semestre, en el qual desenvoluparà plenament un senzill processador, *LittleProc*.

0.2. Assignatura i estudiants *target*

La metodologia d'ensenyament desenvolupada per nosaltres ha estat aplicada durant quatre anys majoritàriament sobre estudiants de segon any de la titulació d'enginyeria informàtica en una assignatura centrada en el disseny de sistemes digitals. Aquesta assignatura és presencial, de durada semestral, i té assignats 6,5 crèdits ECTS, en què es dedica un 20% del temps a la teoria, un 40% als seminaris de problemes i el 40% restant a l'apartat de pràctiques.

El retrat robot dels estudiants que han seguit la nostra metodologia docent està definit per les característiques següents. Són estudiants de segon any que ja tenen un cert *background* en l'àmbit dels sistemes digitals, adquirit en assignatures del primer any del grau. En general, tenen assimilats els fonaments de la lògica booleana, coneixen els blocs combinacionals i seqüencials bàsics, i tenen certes nocions sobre aquests darrers. A més també coneixen el comportament funcional d'un processador elemental. El nivell de profunditat i assimilació d'aquests temes pot variar bastant d'un estudiant a un altre.

0.3. Pràctiques: objectius i metodologia global

Les pràctiques van ser dissenyades amb l'objectiu prioritari de possibilitar un coneixement més profund de les màquines algorísmiques complexes, aquelles que tenen una unitat de control microprogramada. Per aconseguir un major grau de motivació en els estudiants, vam establir que no solament dissenyarien i simularien un processador, sinó que, a més, haurien d'arribar a implementar-lo i, en conseqüència, haurien de tenir la possibilitar d'executar-hi un petit programa. Aquesta decisió es va prendre en funció de la nostra experiència docent segons la qual un estudiant se sent molt més atret i implicat per la creació d'un sistema real i tangible que pel desenvolupament d'un sistema solament a escala conceptual. En relació a la implementació, es va considerar que l'opció més viable era un dispositiu programable. En concret, per motius organitzatius interns, es va optar per una FPGA d'Altera.

A més de l'esmentat objectiu prioritari n'hi havia uns altres que s'enumeren a continuació:

— l'aprofundiment dels coneixements relatius a la lògica booleana i els blocs digitals bàsics
— el fet de familiaritzar-se amb alguna eina CAD de disseny electrònic d'ús industrial
— la introducció a la descripció de circuits mitjançant l'ús d'un llenguatge de descripció de *hardware* (HDL), utilitzat a l'entorn professional
— la promoció del treball cooperatiu
— la millora de la capacitat d'expressió oral i escrita
— la capacitat d'organització del treball juntament amb el seu desenvolupament metòdic i continuat al llarg del semestre

Com es pot observar, alguns dels objectius esmentats estan directament vinculats al contingut específic de l'assignatura, mentre que altres fan referència a competències de caràcter transversal.

L'assoliment del conjunt d'objectius indicats i l'avaluació continuada de la seva consecució per part dels estudiants requereix que les pràctiques s'estructurin en sessions de diferents tipus. Així, a més de les típiques sessions de laboratori, vam definir sessions de seminari per introduir conceptes i informació que seran usats posteriorment, i sessions de control on s'avalua el progrés dels estudiants. En la secció següent s'exposa de manera més detallada la planificació i el contingut de totes les sessions.

Aquesta estructuració trimodal de les pràctiques, en les quals se succeeixen els diversos tipus de sessions, comporta un important esforç de coordinació per part del professorat, però alhora també el requereix per part de l'estudiant. Per facilitar aquest procés als estudiants, els professors vam decidir recolzar-nos en Moodle, una aplicació orientada a la gestió de continguts de cursos a través d'un lloc web. Mitjançant Moodle s'indica a l'estudiant, setmana a setmana, el tipus de sessió que realitzarà aquella setmana i la tasca que ha de preparar. A l'apartat de referències bibliogràfiques d'aquest llibre hi ha un enllaç a un servidor de Moodle anàleg al servidor utilitzat amb els nostres estudiants. Un altre aspecte positiu de Moodle és que s'hi pot integrar fàcilment un wiki. Per promoure el treball cooperatiu, nosaltres distribuïm els estudiants en grups de tres membres. Cada grup de treball té el seu wiki incorporat a Moodle i és en aquest wiki on desenvoluparà un seguit d'exercicis escrits al llarg del semestre.

0.4. Estructuració de les sessions

Tenint en compte la càrrega de crèdits de l'assignatura, vam estructurar les pràctiques en 7 sessions de laboratori (de 2 hores cadascuna), 3 sessions de seminari (d'1 h - 1,5 h cadascuna) i tres sessions de control (d'1,5 h). La taula 0.1 mostra la planificació de la totalitat de les sessions de pràctiques.

La primera sessió es dedica bàsicament a què l'estudiant conegui el CAD que utilitzarà al llarg del semestre: Quartus II Web Edition d'Altera. Aquest programari està disponible de manera gratuïta al lloc web del fabricant (www.altera.com). En aquesta primera sessió, els estudiants desenvolupen el primer mòdul de LittleProc (un mòdul força senzill) mitjançant captura d'esquemes. A la segona sessió es desenvolupen dos mòduls bàsics més, també a través de captura d'esquemes.

La sessió 3 està dedicada a impartir uns coneixements bàsics sobre VHDL i mostrar-ne alguns exemples. A la sessió 4 els estudiants desenvolupen quatre mòduls bàsics de LittleProc mitjançant les seves descripcions VHDL prèviament preparades. La sessió 5 es dedica a fer una petita entrevista oral individual a la meitat dels alumnes per verificar el seu grau d'assoliment de coneixements sobre Quartus, el blocs bàsics i VHDL.

A la sessió 6 es presenta el processador LittleProc i es mostra com es pot muntar amb l'ajuda dels blocs bàsics prèviament dissenyats. Les sessions 7 i 8 estan dedicades al disseny de la unitat de procés i a la unitat de control, respectivament. Finalment, la sessió 9 es dedica a fusionar ambdues unitats donant lloc al processador LittleProc. A la sessió 10 es fa una curta entrevista oral individual a l'altra meitat dels estudiants amb preguntes sobre el funcionament de LittleProc.

Sessió	Tipus	Descripció
1	Laboratori	Introducció a Quartus II Web Edition
2	Laboratori	Blocs bàsics (I): captura d'esquemes
3	Seminari	Introducció al VHDL
4	Laboratori	Blocs bàsics (II): disseny amb VHDL
5	Avaluació oral	Preguntes sobre VHDL i blocs bàsics (50 % estudiants)
6	Seminari	Presentació de LittleProc
7	Laboratori	Disseny de la UP
8	Laboratori	Disseny de la UC
9	Laboratori	Disseny de LittleProc
10	Avaluació oral	Preguntes sobre LittleProc (50 % estudiants)
11	Seminari	Introducció a una placa de prototipat d'Altera
12	Laboratori	Implementació de LittleProc a la placa de prototipat
13	Avaluació oral	Preguntes sobre la totalitat de la pràctica (100 % estudiants)

Taula 0.1 Planificació de les sessions

La sessió 11 té com a finalitat fer una introducció a una placa de prototipat d'Altera. Com a colofó de les pràctiques, a la sessió 12 cada grup de treball implementa la seva versió de LittleProc en una placa de prototipat. Finalment a l'última sessió es fa una entrevista oral a tots els estudiants amb preguntes sobre la totalitat de la pràctica.

0.5. Estructura del llibre

El nostre llibre està estructurat de la manera següent. Els primers quatre capítols constitueixen una mena de petit manual de referència de Quartus II Web Edition. En aquests capítols s'afronta des de la seva arrencada inicial i la creació d'un projecte (capítol 1), l'especificació d'un circuit mitjançant captura d'esquemes (capítol 2), la compilació d'un projecte (capítol 3) i la simulació, tant funcional com amb retards, d'un circuit (capítol 4). Al llarg d'aquests quatre capítols es desenvolupa a mode d'exemple el primer mòdul de LittleProc. Aquests capítols són seguits durant la sessió 1 i són útils en la resta de sessions de laboratori com a manual de referència de Quartus. El capítol 5 fa un recordatori sobre les diferències entre les baldes i els *flip-flops* i presenta alguns temes interessants com la connectivitat per nom i la realització d'esquemes compactes. Al final del capítol es troben les especificacions de dos nous mòduls senzills de LittleProc, els quals han de ser desenvolupats per part dels estudiants a la sessió 2 mitjançant captura d'esquemes.

El capítol 6 és una petita introducció a VHDL i conté un bon nombre d'exemples il·lustratius. La part final del capítol conté les especificacions de quatre mòduls bàsics que els estudiants han de desenvolupar a la sessió 4 mitjançant l'ús de VHDL.

Els capítols 7 - 12 presenten el disseny de LittleProc en la seva versió de 2 busos. El capítol 7 està centrat en el disseny de la UP i serveix als estudiants com a referència per fer la UP durant la sessió 7. Els capítols 8 - 10 estan dedicats principalment a la unitat de control, la ROM de microprogramació i el conjunt d'instruccions de LittleProc. El seu contingut regeix el desenvolupament de la UC per part dels estudiants en la ses-

sió 8. Per la seva banda, els capítols 11 i 12 estan dedicats a la RAM i a l'assemblatge final de LittleProc. Els estudiants els segueixen durant la sessió 9.

El capítol 13 està dedicat íntegrament a presentar com seria LittleProc en la seva versió d'un bus. En el capítol es mostren les diferències que hi hauria en relació amb la versió de dos busos. En la nostra pràctica docent anem alternant les dues versions en cursos successius. Un any els estudiants fan la versió de dos busos, l'any següent la versió d'un bus, i així successivament.

Els capítols 14 i 15 estan dedicats a la implementació en placa de LittleProc. En aquests capítols es fa una introducció a dues plaques de prototipat d'Altera, concretament la UP3 Education Kit (capítol 14) i la DE2 (capítol 15). En cada capítol també s'explica quines modificacions s'han de fer a LittleProc per facilitar la seva implementació sobre la placa descrita al capítol. El contingut d'aquests capítols serveix de referència a la tasca que duen a terme els estudiants a la sessió 12 quan *baixen* a placa la seva versió de LittleProc.

Finalment el capítol 16 conté un recull de propostes d'exercicis escrits i preguntes orals. Els primers han servit com a base al llarg d'aquests anys per a l'elaboració dels qüestionaris escrits que el grups de treball dels estudiants han hagut de resoldre mitjançant els seus wikis. Per la seva banda, les preguntes orals han estat utilitzades per determinar de manera individualitzada el grau d'assimilació de coneixements del estudiants durant la realització de les sessions d'avaluació oral.

1. Primeres passes

Arrenqueu el programari Quartus II Web Edition, fent un doble clic sobre la icona que trobareu a l'escriptori de Windows o bé amb l'ordre

`Iniciar → Tots els programes → Altera → Quartus II (versió) Web Edition → Quartus II (versió) Web Edition`

Automàticament s'obre l'àrea de treball de Quartus II Web Edition (Fig. 1.1). Hi podeu apreciar les diverses finestres que donen accés a totes les utilitats de Quartus II Web Edition (d'ara endavant direm per simplificar, Quartus). Mitjançant el ratolí el dissenyador estableix en quina finestra vol treballar. Si s'obre la finestra *Getting Started With Quartus II Software* es procedirà a tancar-la.

La majoria de les ordres de Quartus poden ser executades mitjançant un conjunt de menús que es troben situats sota la barra del títol. Per exemple, si feu un clic amb el botó esquerre del ratolí sobre l'entrada anomenada **`File`** es desplega un menú vertical (Fig. 1.2) on podeu observar un bon nombre d'ordres relacionades amb la manipulació de fitxers.

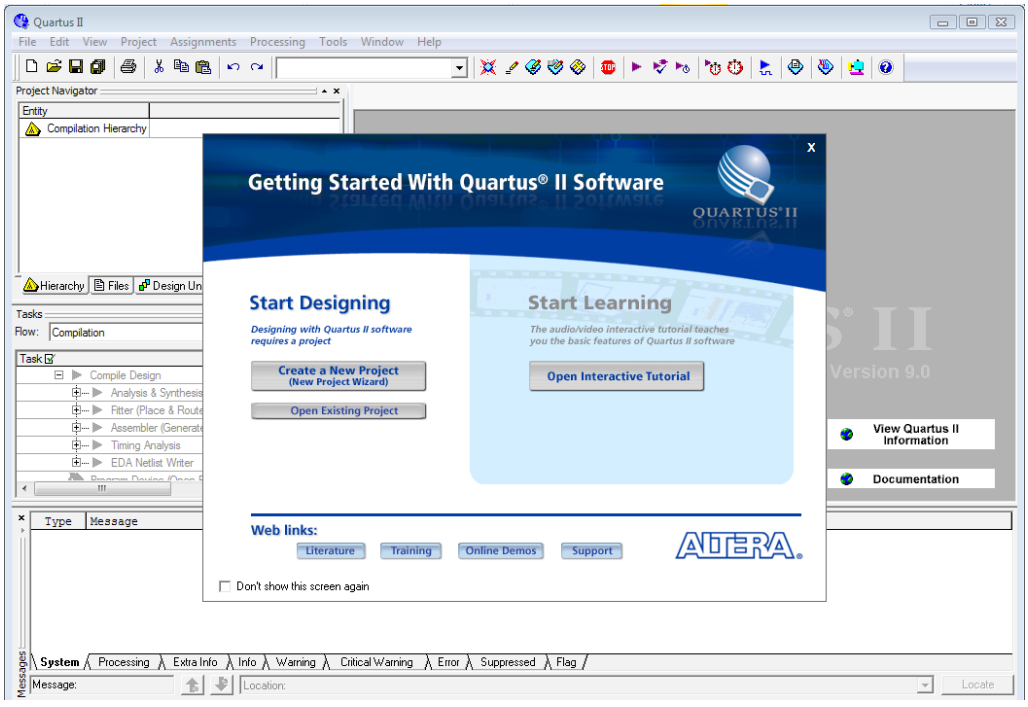


Fig. 1.1 Obertura de Quartus II Web Edition

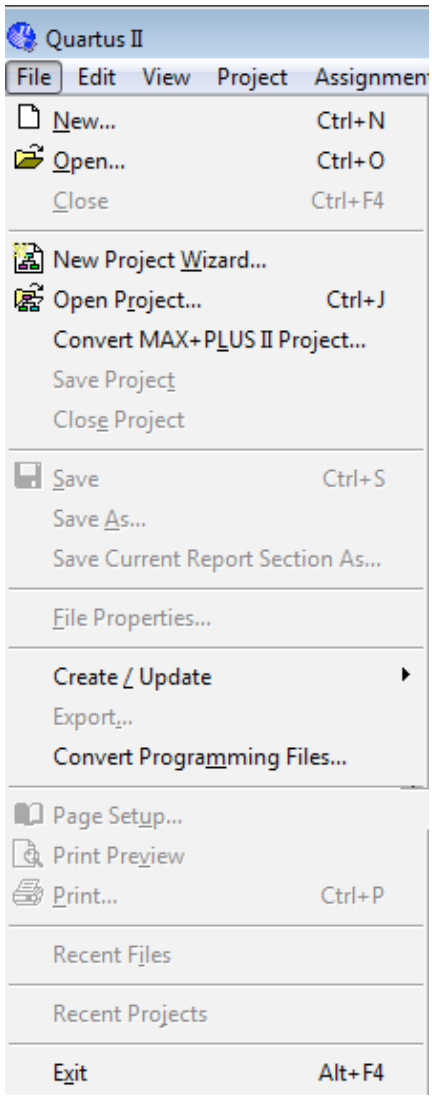


Fig. 1.2 Menú vertical d'ordres d'obertura i tancament de fitxers i projectes

Algunes ordres tenen associada una *bindkey* (una tecla o grup de tecles que en ser premudes provoquen l'execució d'aquesta ordre). Quan una ordre té associada una *bindkey*, aquesta apareix a la seva dreta al menú. Per exemple, Crtl+N és la *bindkey* de l'ordre **`New`**.

1.1. Ajuda en línia de Quartus

Quartus té incorporada una documentació en línia que respon moltes de les preguntes o els dubtes que es poden produir durant el procés de disseny. A tota la documentació es pot accedir des del menú activable mitjançant l'opció **`Help`**. Per exemple, si el dissenyador vol ajuda sobre com utilitzar l'ajuda hauria d'executar:

`Help → How to Use Help`

A més a més, l'usuari pot buscar ràpidament ajuda sobre termes o conceptes mitjançant l'ordre:

Help → Search

Aquesta ordre provoca l'obertura d'un formulari en el qual l'usuari introdueix el terme sobre el qual vol informació.

Una altra manera d'obtenir ajuda de manera ràpida sobre una determinada aplicació de Quartus és prémer la tecla **F1**, la qual provoca l'obertura d'una finestra d'ajuda amb les ordres vinculades a aquesta aplicació.

1.2. Creació d'un nou projecte

A Quartus qualsevol circuit o subcircuit que està sent desenvolupat per un dissenyador rep el nom genèric de *projecte de disseny*. Quartus emmagatzema tota la informació d'un projecte en un únic directori prèviament establert pel dissenyador.

El primer disseny que es desenvolupa és un circuit molt senzill: el bloc ctrlSeq que implementa tres funcions booleanes i que després és utilitzat en la realització del processador LittleProc.

Per començar a treballar en un nou disseny el primer pas és crear un nou *projecte de disseny*. Quartus II facilita aquesta tasca mitjançant la utilització d'un assistent. Per invocar-lo s'executa l'ordre:

File → New Project Wizard

Automàticament s'obre una finestra amb finalitat merament informativa (Fig. 1.3). Marqueu la casella **Don't show me this introduction again** perquè les properes vegades aquesta informació no sigui mostrada. Premeu el botó **Next**.

A continuació apareix el primer formulari de l'assistent (Fig. 1.4). Hi heu d'introduir el nom del directori on s'emmagatzemaran tots els fitxers del nou projecte, el nom del projecte i el nom de la *cel·la principal* (*top level entity*) del nostre disseny. Per cel·la principal entenem la cel·la que es trobaria en el nivell superior de jerarquia del nostre disseny. En el formulari heu d'introduir la informació següent (on posa *Grupx* i *Subgrupx* heu de posar el vostre grup de pràctiques i subgrup de treball, respectivament):

What is the working directory for this project? :
D:\DSD\Grupx\Subgrupx\LittleProc

What is the name of the project? :
LittleProc

What is the name of top-level entity in your project?...:
ctrlSeq

L'última pregunta fa referència a la *top level entity*. Tal com hem dit, aquesta *entity* és el bloc que es trobaria en el nivell superior de jerarquia del disseny. De moment es posa el nom del primer bloc que es realitza (ctrlSeq) però, més endavant, a mesura que avanci el disseny, la *top level entity* canvia. Hi ha una ordre (**Project → Set as Top-Level Entity**) que permet definir en cada moment quina és la *top-level entity*.

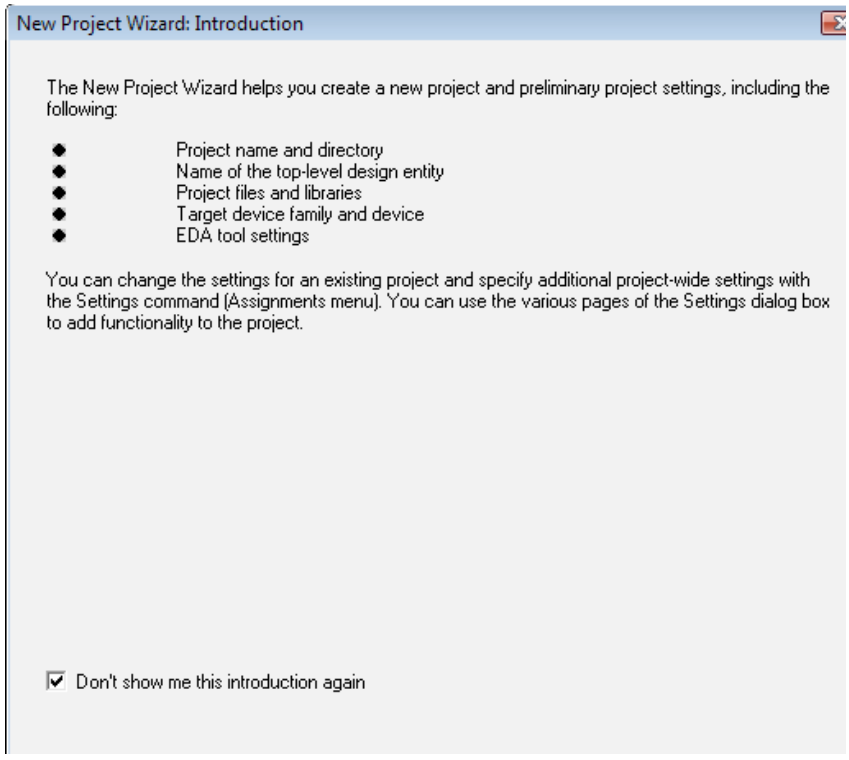


Fig. 1.3 Creació d'un nou projecte: finestra inicial informativa

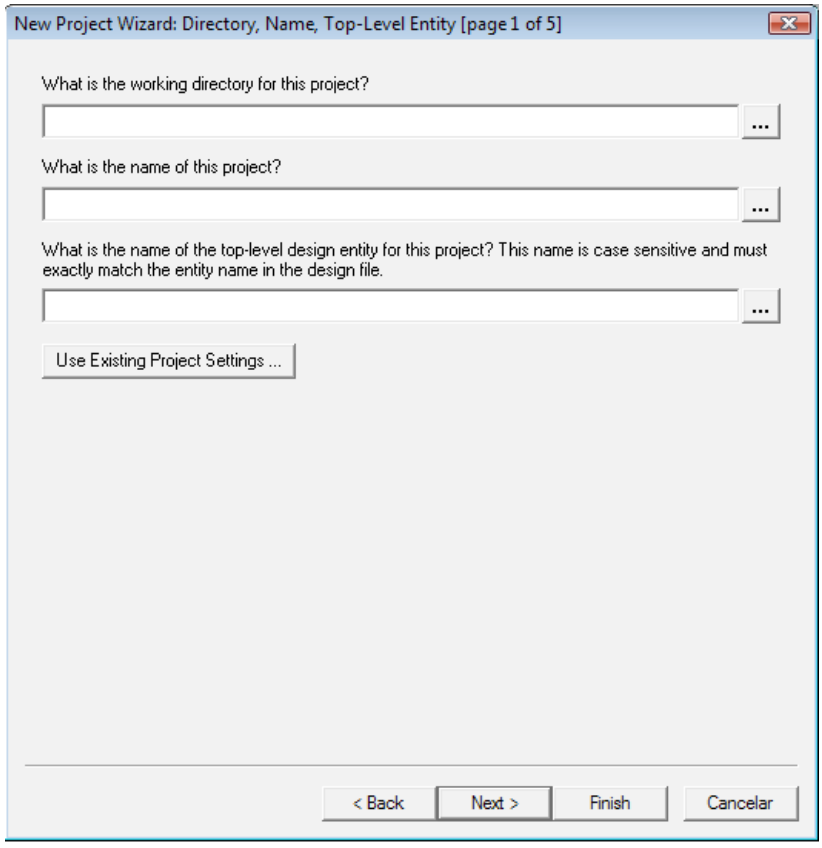


Fig. 1.4 Creació d'un nou projecte: introducció del directori, nom i cel·la principal

Finalment heu de prémer el botó **Next**. Com que el directori que heu introduït no existeix, una finestra s'obre demanant-vos la confirmació per a la seva creació. Doneu-hi el vistiplau prement el botó **Yes**.

Automàticament s'obre el segon formulari de l'assistent (Fig. 1.5) que permet al dissenyador incloure fitxers amb informació sobre subcircuits ja creats, útils per a l'elaboració del nou disseny. El disseny de ctrlSeq no requereix cap subcircuit, per la qual cosa heu de deixar tots el camps buits i pitjar el botó **Next**.

El tercer formulari permet que el dissenyador trií el dispositiu, és a dir, el model de FPGA, en el qual es vol implementar el disseny. Haureu d'escollir una de les dues opcions següents en funció de la placa que s'utilitzarà per a la implementació:

- família de dispositius **Cyclone** i dispositiu **EP1C12Q240C8** si la placa és l'Altera UP3 Education Kit (Fig. 1.6)
- família de dispositius **Cyclone II** i dispositiu **EP2C35F672C6** si la placa és l'Altera DE2 (Fig. 1.7)

Finalment premeu el botó **Next**.

El quart formulari de l'assistent (Fig. 1.8) permet al dissenyador especificar eines de CAD d'altres empreses informàtiques que es volen utilitzar per a certes tasques durant el procés de disseny. En el vostre cas, tot el procés de disseny es fa íntegrament amb Quartus, per la qual cosa podeu deixar tots els camps en blanc. Simplement premeu el botó **Next**.

Finalment s'obre una finestra (Fig. 1.9) on s'observa un resum de tota la configuració establerta mitjançant els formularis anteriors. Si detecteu que heu comès un error heu de prémer el botó **Back** per anar endarrere i corregir l'error. En cas contrari, heu de prémer el botó **Finish** i el procés de creació del nou projecte haurà acabat.

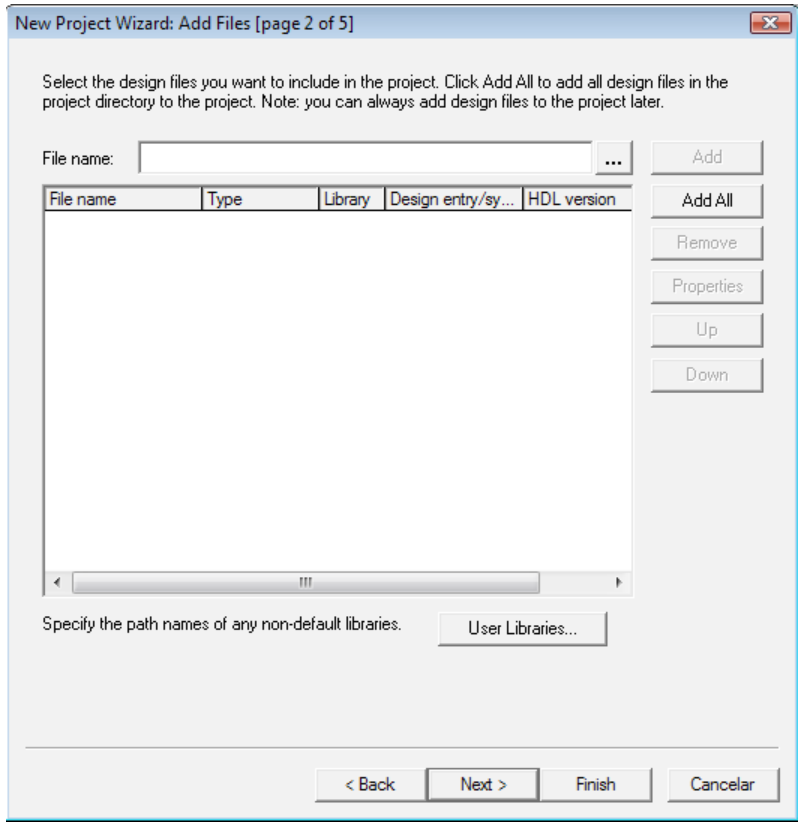


Fig. 1.5 Creació d'un nou projecte: addició de fitxers al projecte

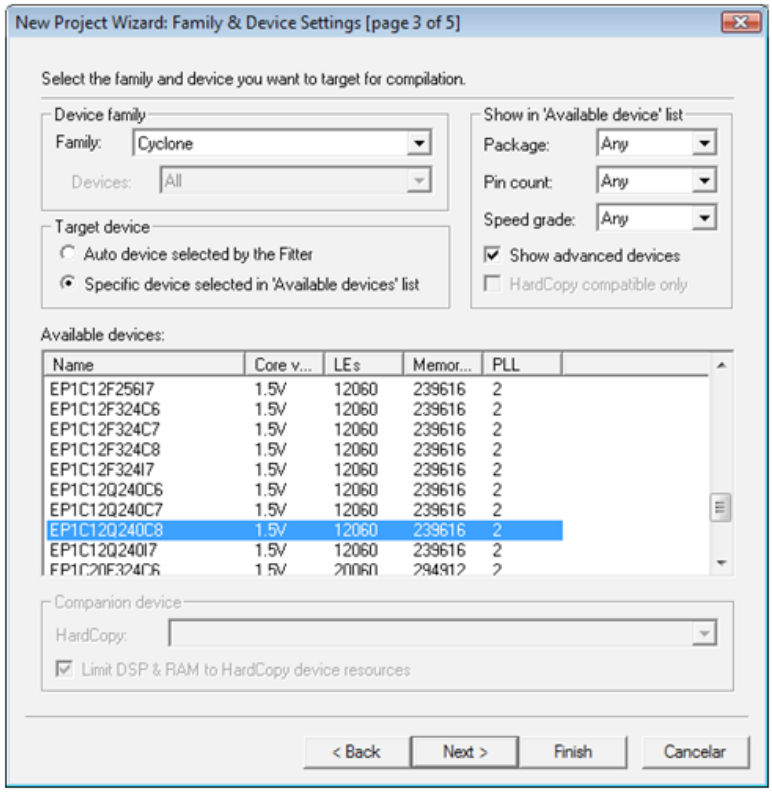


Fig. 1.6 Creació d'un nou projecte: especificació del dispositiu EP1C12Q240C8

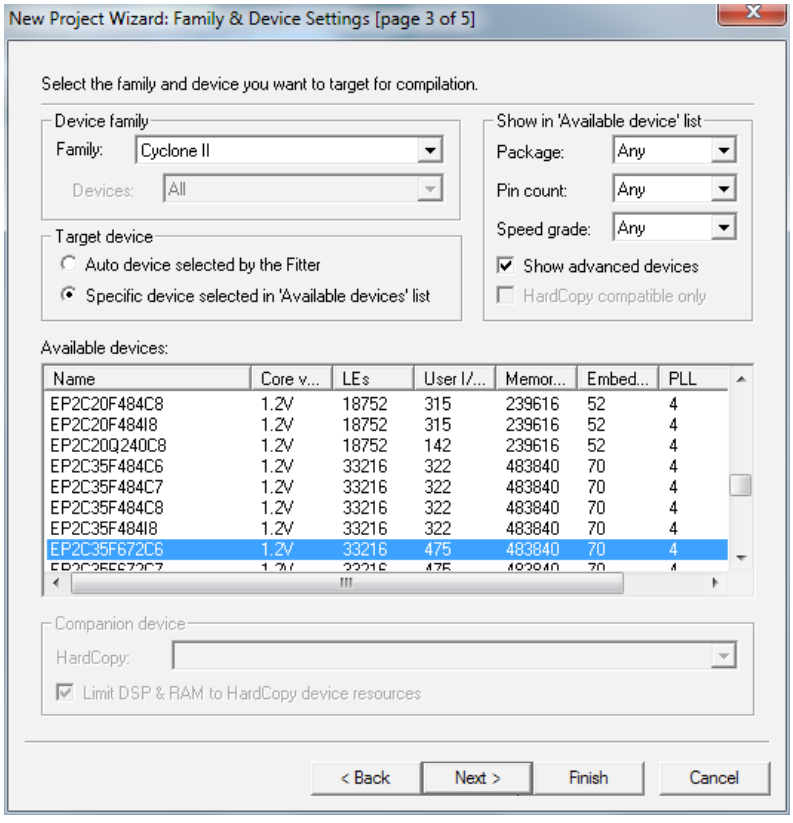


Fig. 1.7 Creació d'un nou projecte: especificació del dispositiu EP2C35F672C6

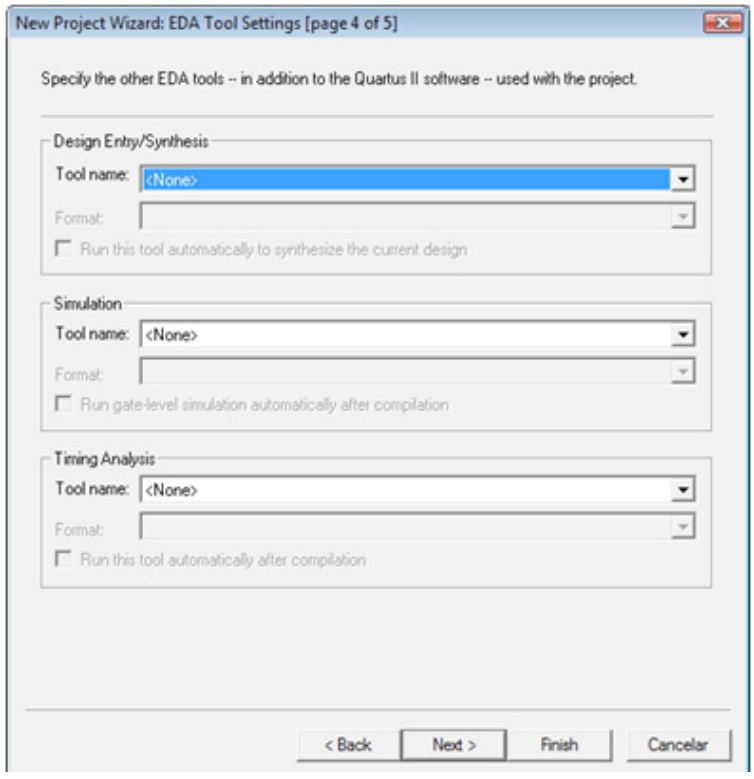


Fig. 1.8 Creació d'un nou projecte: especificació d'eines CAD addicionals

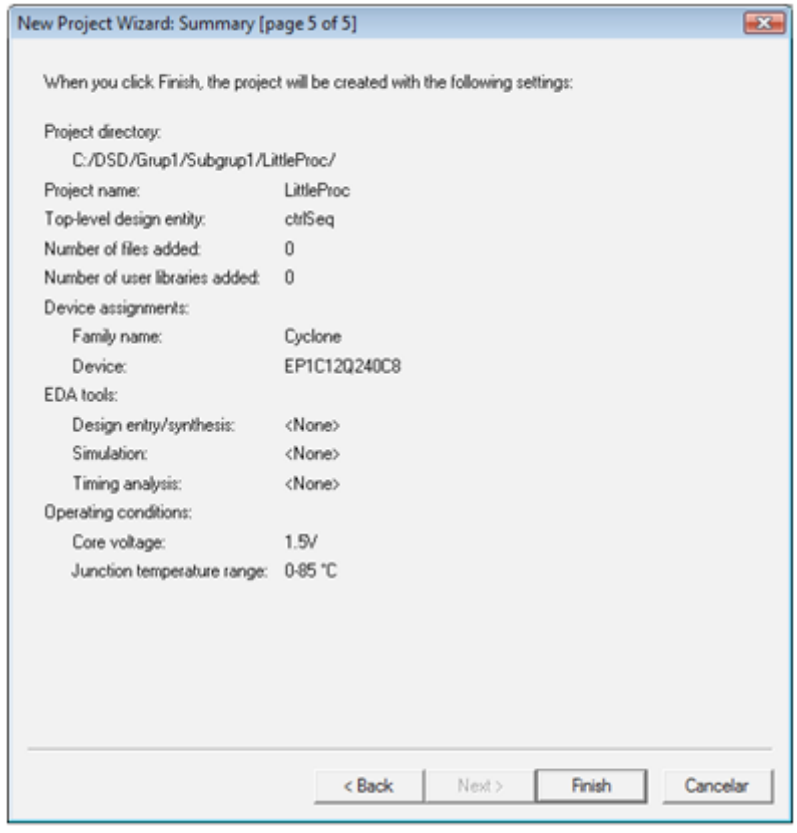


Fig. 1.9 Creació d'un nou projecte: finestra resum del projecte

2. Introducció del disseny mitjançant captura d'esquemes

2.1. Invocació de l'editor d'esquemes de Quartus

Una de les maneres possibles d'introduir un disseny és mitjançant la captura d'esquemes, procés en el qual s'especifica l'esquema lògic del disseny. En el vostre cas heu d'especificar l'esquema lògic de la cel·la ctrlSeq. Aquesta cel·la és una cel·la combinacional de tres entrades i tres sortides, totes elles d'un bit. Les entrades són T1, T0 i g. Les sortides són W1, W2 i W3. Les funcions a implementar són les següents:

$$W1 = \overline{T1} \cdot \overline{T0} \cdot g + \overline{T1} \cdot T0$$
$$W2 = T1 \cdot \overline{T0}$$
$$W3 = T1 \cdot T0$$

A la Taula 2.1 podeu observar la taula de veritat de la cel·la ctrlSeq i a la Fig. 2.1, el seu esquema lògic.

g	T0	T1	W1	W2	W3
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	0	0	1
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	1	0	0
1	1	1	0	0	1

Taula 2.1 Taula de veritat de ctrlSeq

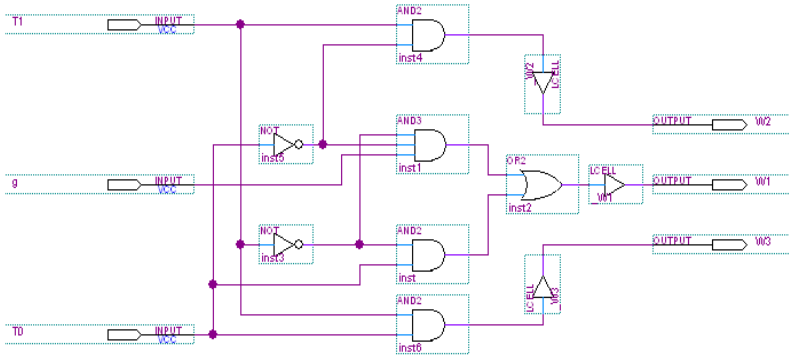


Fig. 2.1 Esquema lògic de ctrlSeq

Per crear un esquema, utilitzeu l'editor d'esquemes de Quartus, el qual s'invoca executant:

File → New (*bindkey* **Ctrl+N**)

S'obre un formulari (Fig. 2.2) en el qual heu d'escollir **Design Files → Block Diagram / Schematic File** i a continuació prémer el botó **OK**. Automàticament es produeix l'obertura de la finestra corresponent a l'editor d'esquemes (Fig. 2.3).

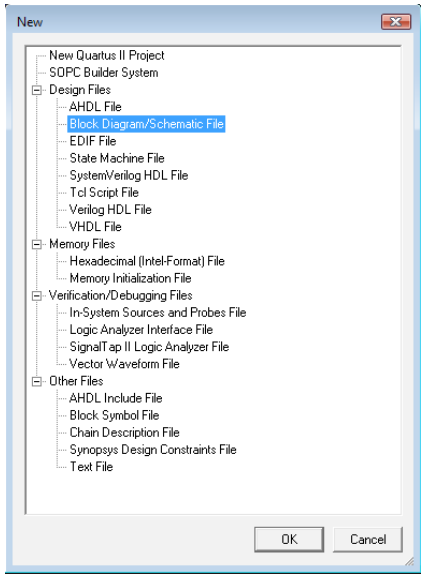


Fig. 2.2 Formulari per a la creació d'un nou esquema

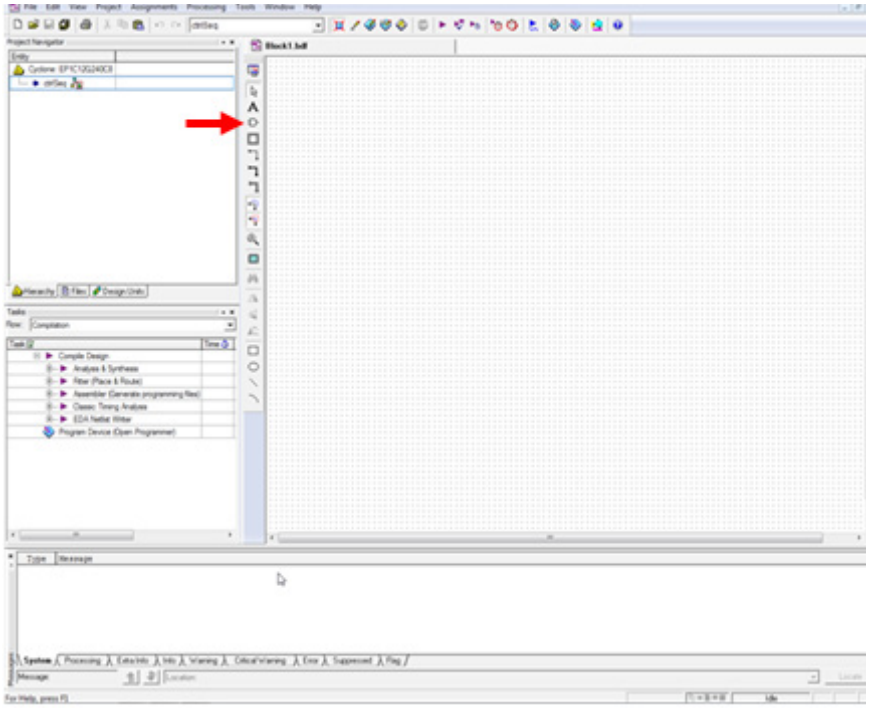


Fig. 2.3 Finestra de l'editor d'esquemes

2.2. Addició de portes lògiques

L'editor d'esquemes proporciona diverses biblioteques que contenen portes lògiques que poden ser utilitzades en la realització d'un disseny. En el vostre cas heu d'utilitzar una biblioteca que s'anomena *primitives*. Hi ha tres mètodes diferents per incloure una porta en un disseny:

Mètode 1

Feu un doble clic amb el botó esquerre del ratolí sobre el fons en blanc de la finestra de l'editor (no sobre cap component prèviament ubicat a la finestra d'edició). Automàticament s'obre una finestra (Fig. 2.4). La finestra anomenada *Libraries* conté diverses biblioteques entre les quals hi ha la biblioteca *primitives*. Expandiu les carpetes fins a trobar la llista de portes lògiques pertanyent a la biblioteca *primitives*. Feu un doble clic a la porta que voleu incorporar a l'esquema del vostre disseny.

Mètode 2

Executeu l'ordre **`Edit → Insert Symbol`**. Automàticament s'obre la mateixa finestra citada en el mètode 1. Seguint el mateix procés descrit en el mètode 1 podeu inserir una porta en el vostre esquema.

Mètode 3

Feu un clic amb el botó esquerre del ratolí sobre el petit símbol corresponent a una porta AND del menú vertical de botons (vegeu el botó indicat per una fletxa a la Fig. 2.3). Això provoca que s'obri la finestra ja citada en els mètodes anteriors.

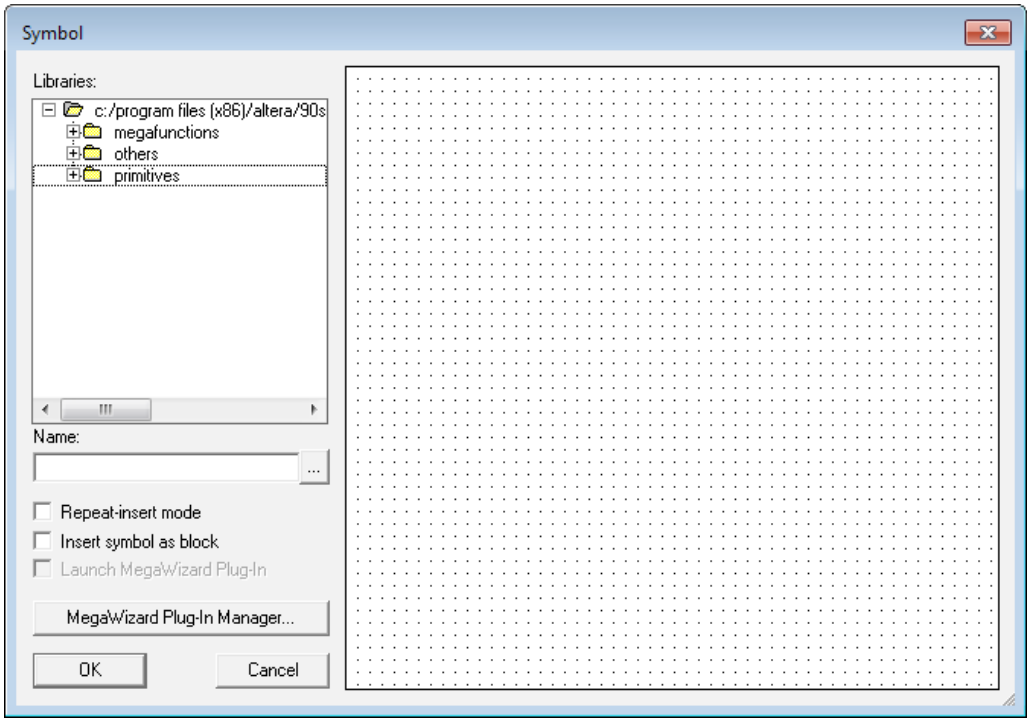


Fig. 2.4 Addició d'un nou símbol

Podeu utilitzar qualsevol dels mètodes anteriors per inserir en el vostre esquema les deu portes que constitueixen la cel·la **ctrlSeq**. Us recomanem que els utilitzeu tots tres.

Si voleu posar més d'una instància de la mateixa porta marqueu l'opció **`Repeat-insert mode`**, la qual us permet posar tantes instàncies com necessiteu. Per finalitzar la introducció d'instàncies del mateix símbol premeu la tecla **`Esc`**.

2.3. Operació amb símbols (selecció, moviment i rotació)

En un esquema, qualsevol símbol d'una porta pot ser seleccionat. Per seleccionar un símbol, situeu-vos amb el ratolí sobre el símbol i feu-hi un clic amb el botó esquerre. El símbol seleccionat queda ressaltat en color blau.

De vegades pot interessar seleccionar un conjunt de símbols. En aquest cas es pot fer una selecció incremental afegint cada vegada un nou símbol al conjunt de selecció. Per fer aquest tipus de selecció s'ha de prémer la tecla **`Shift`** mentre es fa un clic amb el botó esquerre del ratolí sobre el símbol que es vol afegir al conjunt d'elements seleccionats. Si es vol treure un element del conjunt de selecció s'ha de seguir exactament el mateix procés fet per afegir-lo, és a dir, utilitzant la tecla **`Shift`**.

Una altra possibilitat per seleccionar un conjunt de símbols és fer una selecció per àrea, delimitant amb el ratolí una zona rectangular de selecció. Per realitzar-la, es fa un clic amb el botó esquerre del ratolí en un vèrtex del rectangle i s'arrossega el ratolí mantenint el botó premut fins al vèrtex contrari.

Per moure un símbol, en primer lloc s'ha de seleccionar, i després arrossegar-lo amb el ratolí fins a la posició desitjada. Per facilitar la precisió en el posicionament dels símbols pot ser útil activar la visualització d'una graella de línies de referència, la qual cosa s'aconsegueix mitjançant l'ordre:

`View → Show Guidelines`

Si us interessa rotar un símbol (ho necessitareu amb alguns *buffers* de ctrlSeq), heu de seleccionar-lo i executar l'ordre:

`Edit → Rotate by Degrees → nombre de graus`

En finalitzar el procés d'addició de símbols el vostre esquema ha de ser similar al mostrat a la Fig. 2.5. Les cel·les LCELL són *buffers* que es posen per motius tècnics que seran explicats en capítols posteriors.

2.4. Addició de terminals (*ports*) d'entrada i sortida

Una vegada que ja heu introduït totes les portes lògiques, heu d'afegir els terminals (*ports*) d'entrada i sortida que representen les entrades i sortides de **ctrlSeq**. Els terminals s'afegeixen seguint el mateix sistema que heu seguit per introduir els símbols de les portes. Els trobareu dins la carpeta *pin* que es troba dins la biblioteca *primitives*. N'heu d'agafar tres de tipus *input* (per als senyals d'entrada) i tres de tipus *output* (per als senyals de sortida).

Per donar un nom a cadascun dels terminals, feu un doble clic sobre la paraula *pin_name* del terminal al qual voleu donar un nom. El nom del terminal queda selecci-

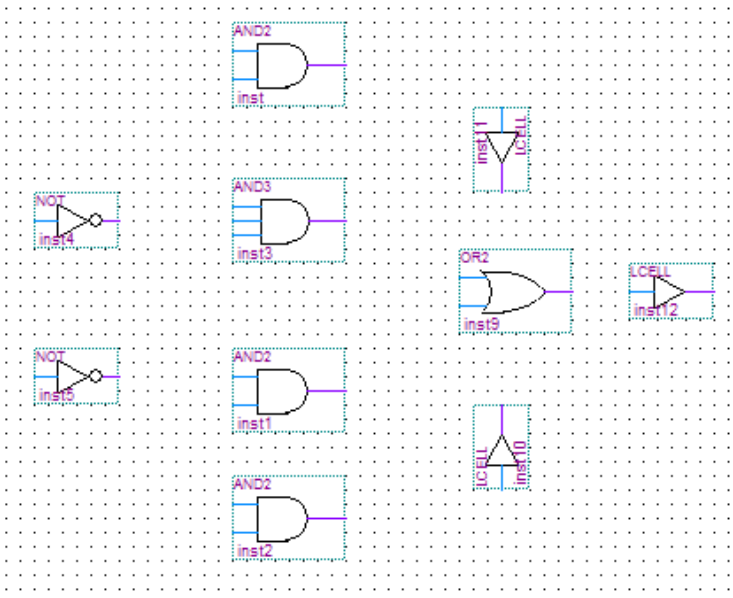


Fig. 2.5 Posicionament de cel·les

onat i aleshores pot ser editat. Poseu als ports d'entrada els noms T1, g i T0, i als ports de sortida, el noms W2, W1 i W3.

En finalitzar el procés d'addició de terminals el vostre esquema ha de ser similar al presentat a la Fig. 2.6.

2.5. Addició de línies de connexió

El següent pas és afegir les línies per connectar les portes (i els terminals) entre si. Per posar una línia de connexió d'un bit heu de fer un clic sobre la sisena icona del menú vertical (*orthogonal node tool*). A continuació feu un clic amb el botó esquerre del ratolí sobre el punt d'inici de la connexió i, mantenint el botó premut, desplaceu el ratolí fins a arribar a l'altre extrem de la connexió. Com a resultat de l'operació es crea una línia que connecta elèctricament els elements que es troben en ambdós extrems.

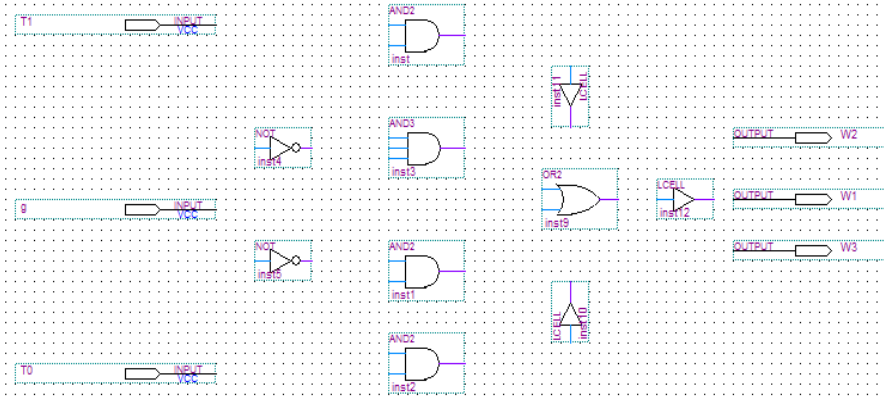


Fig. 2.6 Posicionament de pins

De la mateixa manera, per posar un bus heu de fer un clic sobre la setena icona del menú vertical (*orthogonal bus tool*).

Podeu observar que quan un dels extrems és un punt d'una línia de connexió prèviament dibuixada, es crea un punt gros indicador d'una connexió elèctrica.

Si us interessa apropar la imatge podeu utilitzat una ordre de zoom. Hi ha diverses ordres de zoom que estan agrupades a l'entrada de menú **`View`**. També hi ha una icona al menú vertical en forma de lupa (onzena icona del menú vertical) que permet ampliar una zona concreta del disseny. Per utilitzar-la, feu un clic amb el botó esquerre del ratolí sobre la icona i, a continuació, amb el ratolí marqueu la zona rectangular que voleu ampliar. Veureu que la zona marcada s'engrandeix i ocupa quasi la totalitat de la finestra d'edició d'esquemes. Això us facilita la manipulació dels elements que es troben en aquesta zona.

Si voleu tornar a tenir una visió global de tot l'esquema executeu l'ordre:

`View → Fit in Window` (*bindkey* **`Ctrl+W`**)

Quan finalitzeu totes les connexions el vostre esquema hauria de ser similar a l'esquema de la Fig. 2.7. Com que **ctrlSeq** és un circuit molt simple, ha estat molt fàcil dibuixar totes les línies produint un esquema bastant llegible. Quan el circuit és molt més complex, si es dibuixessin totes les línies l'esquema resultant seria incomprensible. En aquests casos s'ha de recórrer a un altre sistema d'especificació de les connexions anomenat *connexió per nom*. Aquest sistema es veurà més endavant.

Proveu ara a moure una porta. Veureu que en moure-la, les línies de connexió «s'estiren» tot mantenint la porta connectada. Això es deu a què, per defecte, l'editor treballa en el mode d'edició anomenat de «*rubberbanding*». Per activar-lo/desactivar-lo, podeu fer un clic sobre la icona de *rubberbanding* (novena icona del menú vertical d'icones). Si el desactiveu, en moure una porta les línies connectades a la porta no queden afectades pel moviment. En conseqüència, tan sols moureu la porta.

Si feu una acció i us en penediu, podeu desfer-la fàcilment executant l'ordre:

`Edit → Undo` (*bindkey* **Ctrl+Z**)

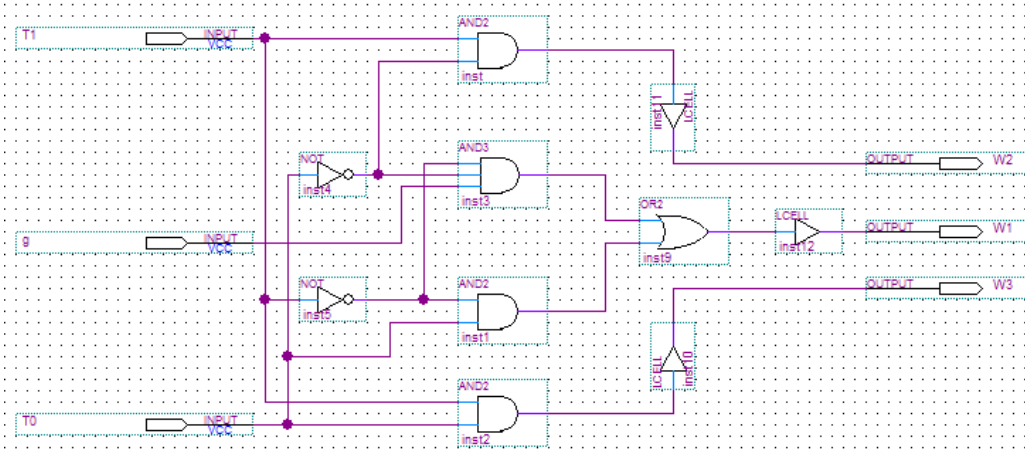


Fig. 2.7 Addició de línies de connexió

2.6. Gravació de l'esquema

Per gravar el vostre esquema, heu d'executar l'ordre:

`File → Save` (*bindkey* **`Ctrl+S`**)

S'obre un formulari en el qual heu d'indicar el nom del fitxer: **ctrlSeq.bdf** .

2.7. Creació d'un símbol i modificació de l'ordre dels seus pins

Per utilitzar la cel·la ctrlSeq en un disseny més complex, es necessita un símbol per fer-hi referència. El símbol es pot crear de manera automàtica mitjançant l'ordre:

`File → Create/Update → Create Symbol Files for Current File`

Com a resultat d'aquesta ordre, Quartus crea un fitxer anomenat ctrlSeq.bsf que conté el símbol de la cel·la. És recomanable que obriu el símbol per comprovar que té els terminals adients. Per fer-ho, heu d'executar l'ordre:

`File → Open` (*bindkey* **`Ctrl+O`**)

Seleccioneu a l'opció "**`Tipus`**" el filtre "**`Graphic Files (*.gdf;*.bdf; *.bsf;*.sym)`**" i òbviament com a nom de fitxer indiqueu **ctrlSeq.bsf**.

Una finestra de l'editor s'obre amb el símbol generat (vegeu el símbol a la Fig. 2.8). Podeu observar que els pins d'entrada i de sortida es troben disposats en el mateix ordre que tenen en l'esquema de la cel·la que heu dissenyat: T1, g, T0 i W2, W1, W3.

A vegades pot interessar modificar l'ordre per defecte dels pins. Com veureu al capítol 8, quan feu servir el símbol de ctrlSeq en l'esquema de la unitat de control de LittleProc, un ordre diferent dels pins del símbol facilitarà el connexionat en l'esquema. L'ordre adient per als pins d'entrada és T1, T0 i g, i per als pins de sortida és W1, W3 i W2.

La manera més senzilla de canviar l'ordre dels pins és canviar els seus noms perquè s'adeqüin a l'ordre desitjat. Per canviar el nom d'un pin, feu doble clic sobre el nom. Al formulari que s'obre, indiqueu el nom que correspongui al camp **`Name`** i premeu el botó **`OK`**. La Fig. 2.9 resumeix tot el procés i la Fig. 2.10 mostra el símbol definitiu amb l'ordre de pins ja canviat.

Finalment, heu de desar el símbol executant **`File → Save`**.

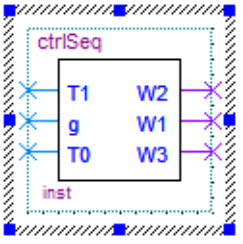


Fig. 2.8 Símbol generat automàticament

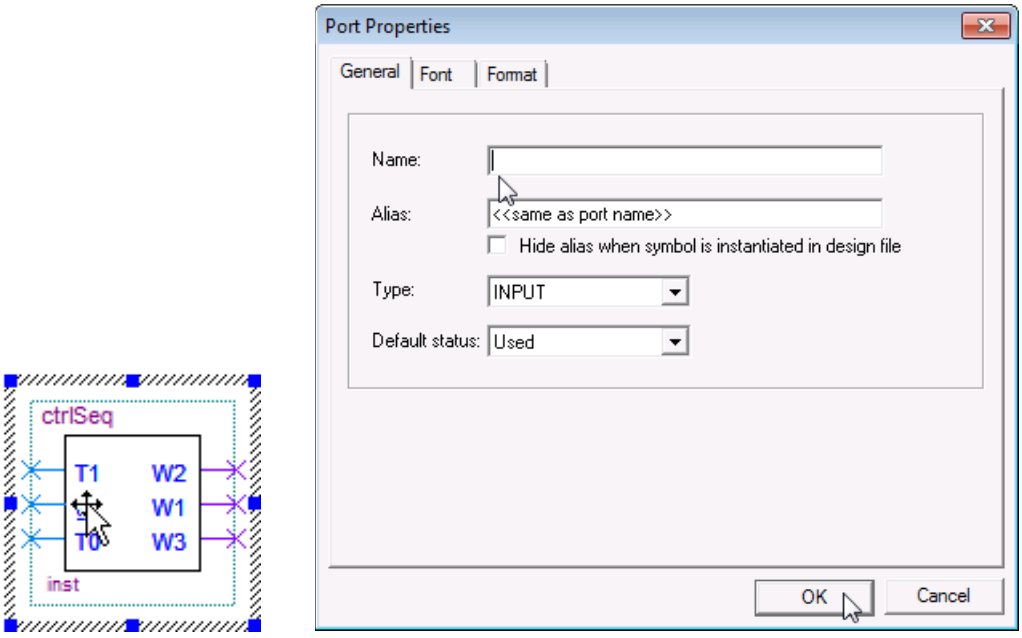


Fig. 2.9 Modificació de l'ordre dels pins en el símbol

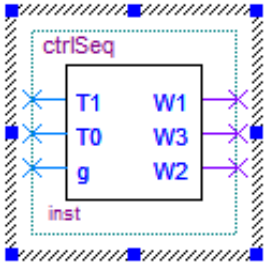


Fig. 2.10 Símbol definitiu de ctrlSeq

2.8. Tancament del projecte

Per tancar el vostre projecte, heu d'executar l'ordre:

`File → Close Project`

2.9. Obertura d'un projecte que ja està creat

Per poder obrir un projecte que ja existeix heu d'executar l'ordre:

`File → Open Project` *LittleProc.qpf* (*bindkey* **`Ctrl+J`**)

D'aquesta manera es torna a tenir actiu el projecte que estàveu implementant i podeu passar a realitzar els passos següents.

3. Compilació

Quan ja heu finalitzat l'esquema podeu començar la compilació i la simulació, que és on es veurà si a l'esquema hi ha errors i si s'han assolit els resultats desitjats.

El primer pas és fer la compilació del disseny realitzat. Això es pot fer de dues maneres diferents. La primera d'elles és prement el botó indicat amb una fletxa a la Fig. 3.1. L'altra possibilitat és mitjançant l'ordre:

`Processing → Start Compilation` (*bindkey* **`Ctrl+L`**)

En iniciar la compilació, a la subfinestra *Tasks* es pot observar el progrés de les diverses etapes de la compilació (Fig. 3.2).

També hi ha l'opció de fer les diferents etapes de la compilació per separat. En aquest cas s'ha d'executar:

`Processing → Compiler Tool`

Automàticament s'obre la finestra mostrada a la Fig. 3.3. Prement els botons assenyalats per les fletxes s'executa cadascuna de les etapes per separat. El botó **`Start`** permet fer la totalitat de la compilació.

Fig. 3.1 Botó de compilació

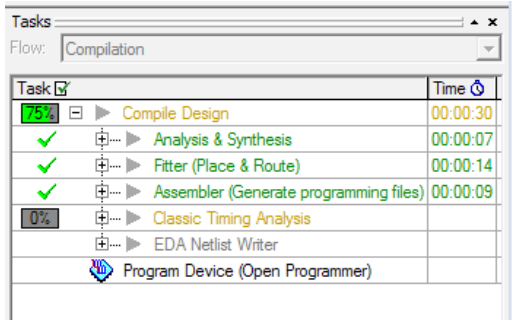


Fig. 3.2 Finestra informativa del progrés de la compilació

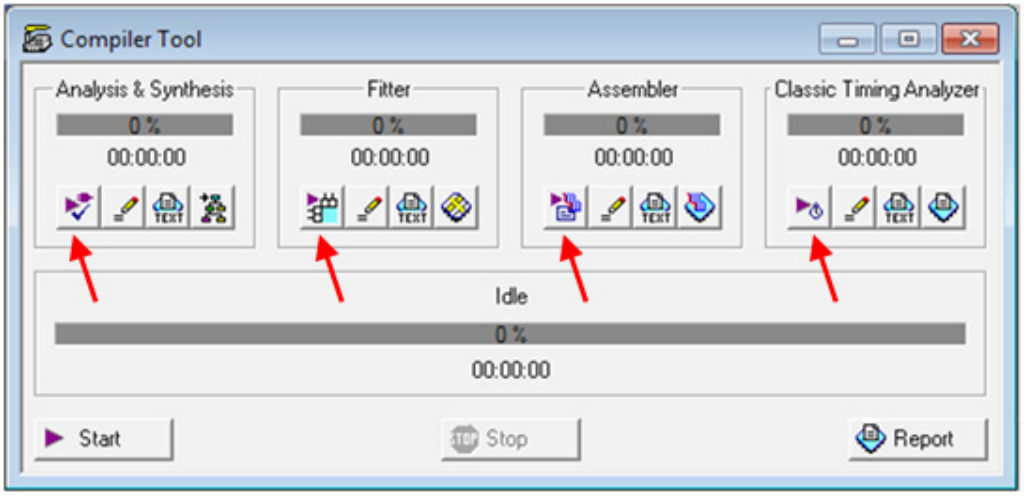


Fig. 3.3 Execució separada de les diverses etapes de la compilació

Durant cada etapa de la compilació, a la part inferior de la pantalla, apareixerà una sèrie d'informacions referents a valors de temps de propagació i capacitats (Fig. 3.4). També poden aparèixer els *warnings*, que són avisos sobre problemes que hi pot haver a l'esquema, i que poden ser l'origen de funcionaments erronis. També és on apareixen els errors (Fig. 3.5), els quals obligatòriament han de ser solucionats per poder prosseguir.

Quan ja està el projecte compilat, és el moment de fer la simulació, on es veu l'evolució dels senyals en el temps, cosa que permet comprovar si s'han assolit els resultats esperats.

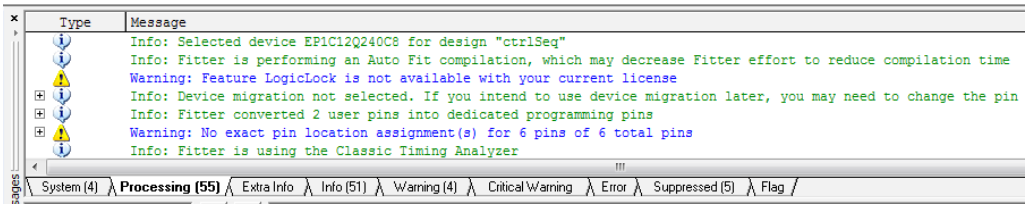


Fig. 3.4 Exemple d'avisos produïts durant la compilació

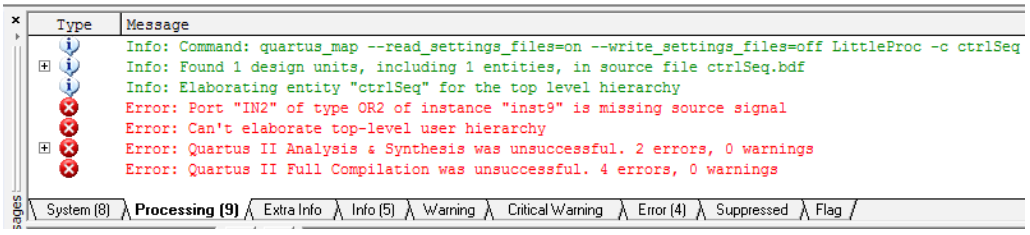


Fig. 3.5 Exemple d'errors produïts durant la compilació

4. Simulació

4.1. Introducció

Abans de començar el procés de simulació s'ha de generar un fitxer de vectors d'entrada que ens permeti comprovar el funcionament correcte del nostre disseny. Cada vector d'entrada està constituït per una sèrie d'estímuls aplicats a les entrades.

En primer lloc s'haurà d'escollir el temps de simulació per defecte que ens interessa: un temps massa llarg alentirà molt el procés, però un temps massa curt farà que no es pugui fer tota la simulació que es desitja. A tal fi, s'executarà la comanda:

`Tools → Options...`

Automàticament s'obre el formulari que es pot observar a la Fig. 4.1. S'escull l'entrada **`Waveform Editor`**. A la finestra associada apareixen una sèrie de camps relatius al temps.

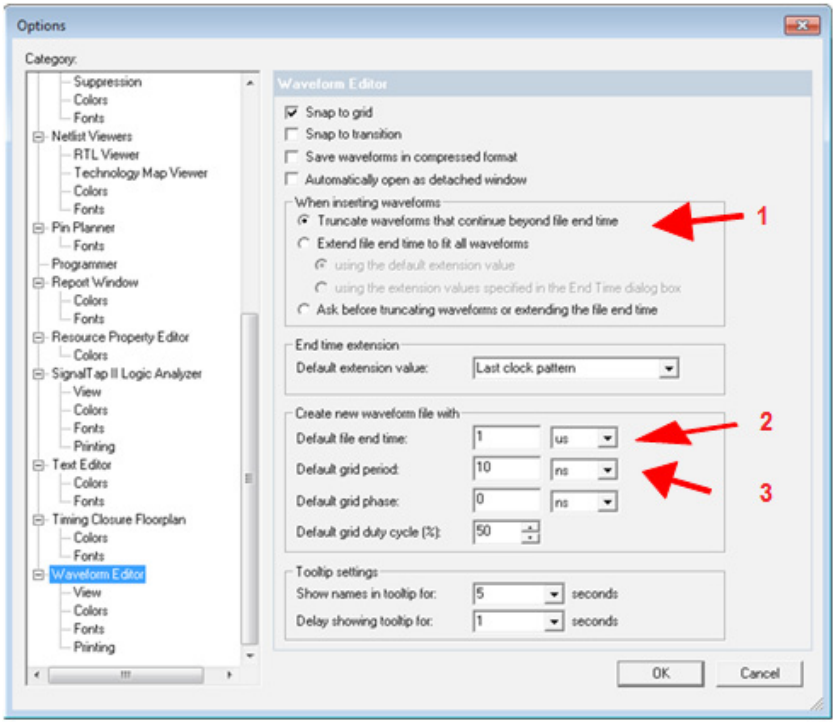


Fig. 4.1 Inicialització de les opcions relatives a les formes d'ona

La primera fletxa (1) assenyala el camp on es determina si la finestra de temps de la simulació és fixa o pot variar en funció dels estímuls. Hi ha tres opcions possibles. La primera opció indica que la finestra de temps és fixa, la segona que depèn dels estímuls, i la tercera que es pregunta abans de variar el temps. S'ha d'escollir la primera opció.

La segona fletxa (2) indica el camp corresponent al temps total per defecte de la simulació i la tercera (3) marca el camp relatiu a l'amplada en temps de la graella d'edició d'ones. Per a aquesta simulació s'utilitza com a temps total per defecte 1μs i com a amplada de graella 10ns, respectivament.

Hi ha dues classes de simulacions: *la simulació funcional*, on només es té en compte l'anàlisi lògica, i la *simulació amb retards*, que intenta apropar-se a la realitat i on es tenen en compte els temps de resposta de les portes així com altres aspectes del sistema. En el nostre cas fem els dos tipus de simulacions: primer expliquem la simulació funcional i després, sobre el mateix disseny, fem la simulació amb retards. A més aprofitem ambdues simulacions per presentar les diverses maneres d'especificar formes d'ona.

4.2. Simulació funcional

4.2.1. Configuració inicial i generació de la netlist

Per indicar que voleu fer una simulació funcional, executeu l'ordre:

`Assignment → Settings...` (*bindkey* **`Ctrl+Shift+E`**)

S'obre un formulari, on s'ha d'escollir l'entrada **`Simulator Settings`**. Apareix la finestra que mostra la Fig. 4.2. En el menú desplegable que assenyala la primera fletxa (1) hi ha tres opcions: **`Functional`**, **`Timing`** i **`Timing using Fast Timing Model`**. S'escull l'opció **`Functional`**.

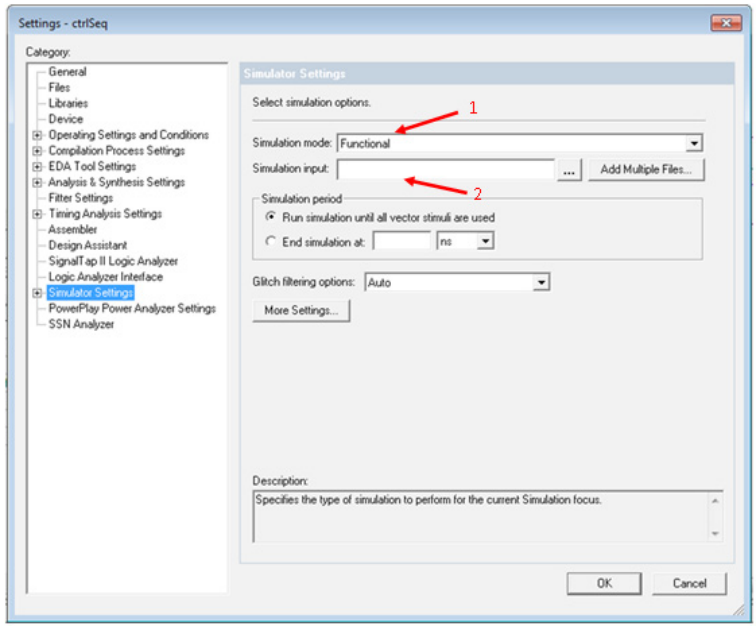


Fig. 4.2 Formulari amb les opcions de simulació (s'ha escollit fer una simulació funcional)

És interessant observar que el camp indicat per la segona fletxa (2) mostra la ruta on està desat el fitxer d'estímuls. Aquest camp de moment pot quedar en blanc. Es finalitza l'ordre prement el botó **OK**.

A continuació cal generar una *netlist* del circuit. Una *netlist* és una llista on figuren els components i els nodes elèctrics del circuit. Per generar la *netlist* s'ha d'executar l'ordre:

Processing → Generate Functional Simulation Netlist

Si no es fa aquest pas, solament es poden fer simulacions tenint en compte el temps de retard. Per tant, és important recordar que per fer la simulació funcional és imprescindible generar aquesta *netlist*.

4.2.2. Introducció dels vectors d'entrada

Abans de començar el procés de simulació s'ha de generar un fitxer de vectors d'entrada que ens permeti comprovar el funcionament correcte del nostre disseny. Cada vector d'entrada esta constituït per una sèrie d'estímuls (formes d'ona) aplicats a les entrades. Una manera de crear el fitxer d'estímuls és executar l'ordre:

Processing → Simulation Debug → Current Vector Inputs

Automàticament s'obre la finestra que presenta la Fig. 4.3. Situant el ratolí sobre el requadre **Name** i prement el botó dret del ratolí, apareix el submenú de la Fig. 4.4. Al submenú esmentat s'escull l'opció

Insert → Insert Node or Bus

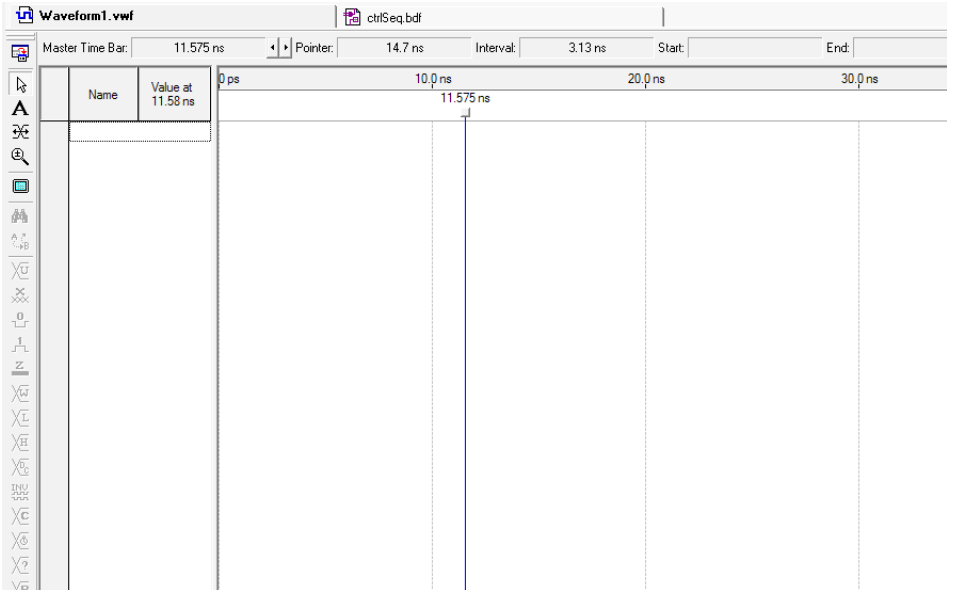


Fig. 4.3 Finestra d'edició de formes d'ona

Quan fem un clic sobre l'opció **`Insert Node or Bus`** apareix la finestra de la Fig. 4.5, en la qual es fa un clic sobre el botó **`Node Finder...`** .

Automàticament s'obre la finestra mostrada a la Fig. 4.6, des de la qual se seleccionen els nodes d'entrada del circuit. La manera més senzilla de fer-ho és seleccionant al camp **`Filter`** l'opció **`'Pins: Input'`** i prement el botó **`List`** (vegeu les fletxes a la Fig. 4.6).

A la columna esquerra de la finestra apareix una llista amb totes les entrades que s'han posat a l'esquema (Fig. 4.7). Per poder-les utilitzar s'han de passar a la columna de la dreta. Per fer-ho hi ha dues opcions: o se seleccionen les entrades que es necessiten i es prem la icona marcada per la fletxa 1, o directament es prem la icona assenyalada per la fletxa 2, la qual arrossega tots els elements de la columna esquerra a la dreta.

Finalment quan ja s'ha seleccionat les entrades per a la simulació (g, T0 i T1 en el nostre cas), es prem el botó **`OK`**.

Retornem a la finestra anterior (Fig. 4.5), on apareix el nom de l'entrada seleccionada o ***`Multiple Items`*** si s'han seleccionat un conjunt d'entrades, amb el format de visualització associat. En aquest exemple, com a **`Radix`** s'escull **`Binary`** i es torna a prémer **`OK`**.

Ara ja tenim les tres entrades que hem posat a l'esquema, però totes estan a '0' (Fig. 4.8). Ens cal aplicar a les entrades els valors que ens interessen. Per augmentar o disminuir l'interval de temps que tenim en pantalla, s'utilitza la icona de la lupa. Situant-nos en l'interval de temps de la pantalla, amb el botó esquerra fem un zoom, i amb el botó dret, ens n'allunyem. També es pot indicar una finestra de temps: situant-nos amb el cursor i arrossegant amb el botó dret enquadrem el tros que ens interessa.

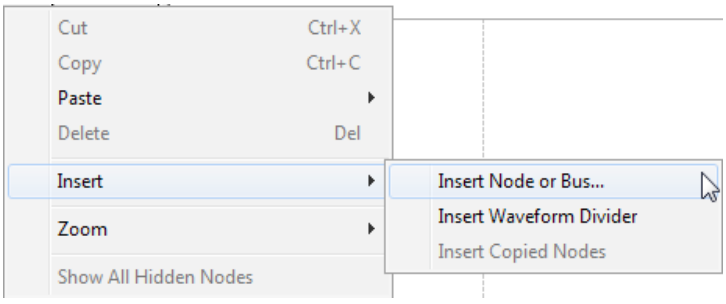


Fig. 4.4 Addició d'entrades a l'editor de formes d'ona (I)

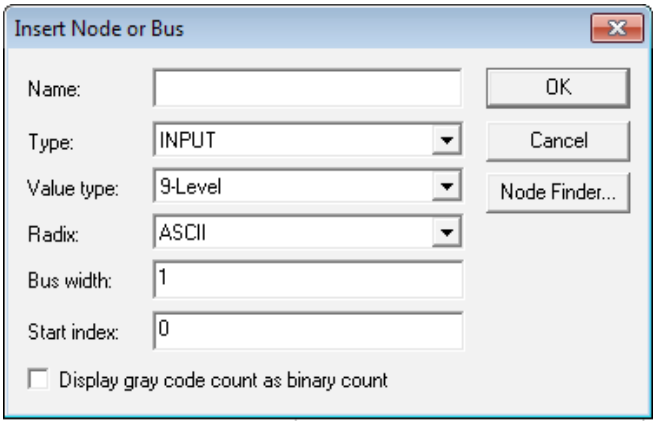


Fig. 4.5 Addició d'entrades a l'editor de formes d'ona (II)

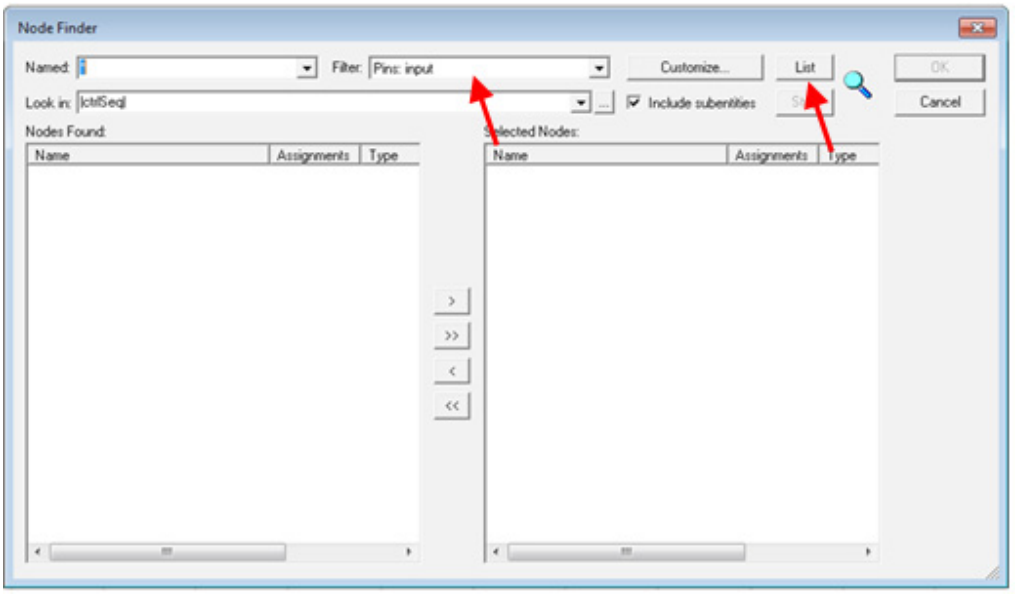


Fig. 4.6 Addició d'entrades a l'editor de formes d'ona (III)

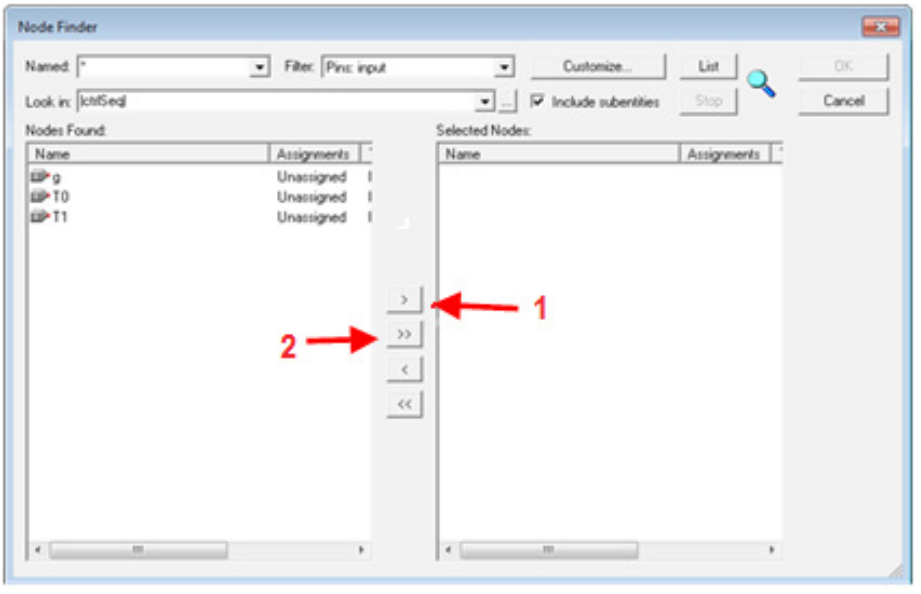


Fig. 4.7 Addició d'entrades a l'editor de formes d'ona (IV)

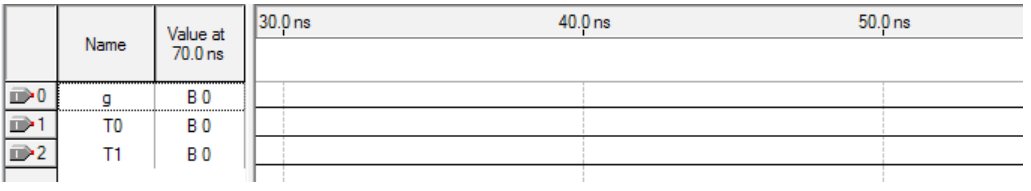


Fig. 4.8 Editor de formes d'ona amb les entrades inicialitzades al valor lògic zero

Quan ja tenim l'interval de temps desitjat, passem a fer les formes d'ona. Seleccionant l'entrada que volem modificar, s'activa el menú de la Fig. 4.9.

Com a estímuls d'entrada per a la simulació poseu tota la taula de veritat de la cel·la ctrlSeq. Això cal fer-ho utilitzant la utilitat de generació de senyals periòdics. Els passos que heu de fer són els següents:

- Seleccioneu l'entrada T1. Premeu la icona del rellotge (element 10 de la Fig. 4.9). Apareix la Fig. 4.10, on heu de deixar els valors que hi ha per defecte. Fixeu-vos que el període és de 10ns. Finalment pitgeu el botó **OK**.

- Seleccioneu l'entrada T0. Premeu la icona del rellotge, poseu al camp **Period** 20ns, i pitgeu el botó **OK**. Això fa que la freqüència del senyal T0 sigui la meitat. Per tant, en un període de T0 se n'han produït dos de T1.

-Repetiu la mateixa operació seleccionant l'entrada g, posant-hi 40ns com a període.

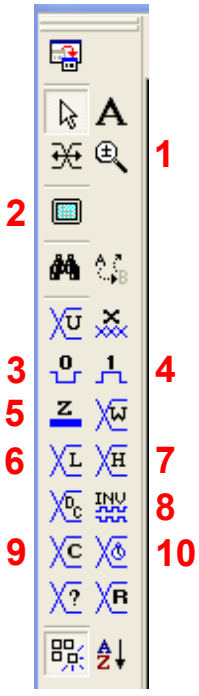
Amb tot això obtenim com a resultat el que es veu a la Fig. 4.11. Com es pot observar, amb un temps de simulació de 40ns n'hi hauria suficient, ja que la resta són repeticions.

Hi ha una altra possibilitat per especificar les mateixes formes d'ona però la veureu més endavant quan feu una simulació amb retards. Aleshores també veureu altres mecanismes de definició de formes d'ona.

Finalment graveu el fitxer amb les formes d'ona creades, executant l'ordre:

File → Save As...

indicant el nom *ctrlSeq.vwf* .



Botó 1: lupa que permet ampliar/disminuir la finestra de temps.
Botó 2: la finestra de simulació ocupa tota la pantalla.
Botó 3: força la zona seleccionada del senyal a '0'.
Botó 4: força la zona seleccionada del senyal a '1'.
Botó 5: força la zona seleccionada del senyal a alta impedància.
Botó 6: força la zona del senyal a '0' feble.
Botó 7: força la zona del senyal a '1' feble.
Botó 8: inverteix la zona del senyal seleccionada.
Botó 9: comptador, va incrementant un compte amb un període determinat.
Botó 10: generació d'un senyal periòdic amb una freqüència determinada.

Fig. 4.9 Botons d'ordres d'edició d'ones

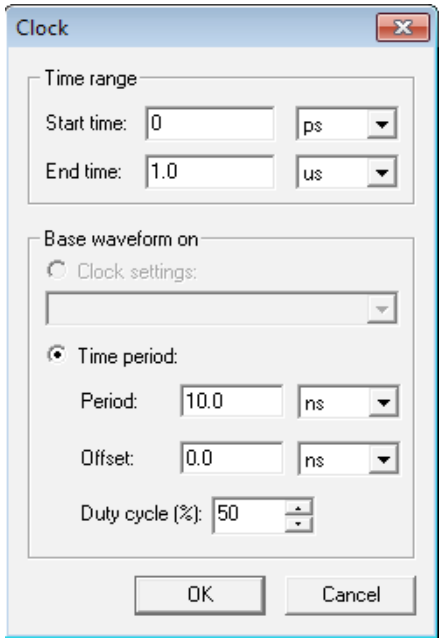


Fig. 4.10 Generació de senyals periòdics

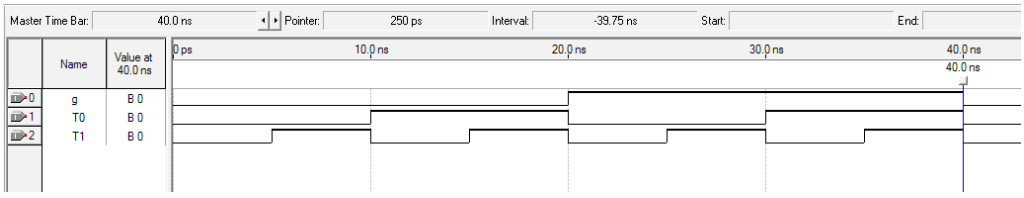


Fig. 4.11 Estímuls d'entrada finalitzats

4.2.3. Realització de la simulació i visualització de resultats

En aquest punt ja sols falta simular. Hi ha dues maneres d'iniciar la simulació. La primera és prement el botó indicat per la fletxa a la Fig. 4.12.

La segona forma consisteix a executar l'ordre:

`Processing → Start Simulation` (*bindkey* **`Ctrl+I`**)

En acabar la simulació s'obre una finestra amb els resultats de la simulació, que són els que es poden veure a la Fig. 4.13. Amb la icona de la lupa, es pot variar l'escala de temps. Si es desitja modificar qualsevol de les entrades d'aquesta simulació, cal tancar aquesta finestra i modificar el fitxer amb els estímuls inicials que es troben al darrere.

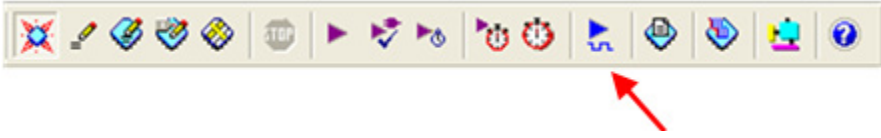


Fig. 4.12 Botó de simulació

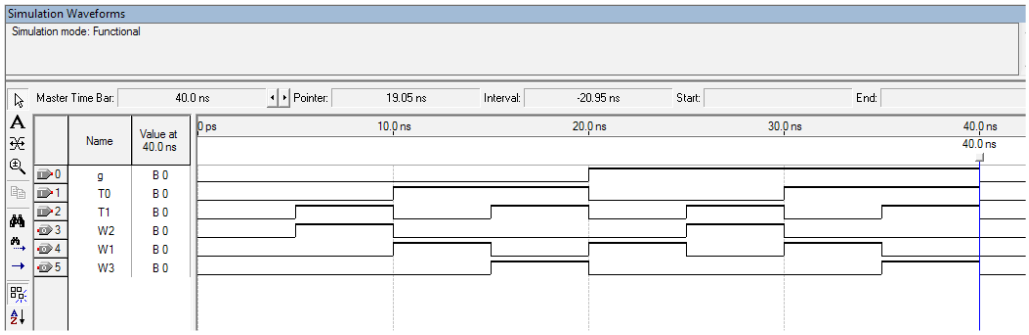


Fig. 4.13 Finestra amb els resultats de la simulació funcional

4.3. Simulació amb retards (*timing simulation*)

Ara repetim la simulació tenint en compte els retards propis dels mòduls o portes utilitzats (*timing simulation*), amb la particularitat que l'especificació dels estímuls es fa mitjançant la definició d'un grup de senyals utilitzant com a període del compte 50ns.

4.3.1. Configuració inicial

Per indicar que es vol fer una simulació amb retards heu d'executar l'ordre:

`Assignment → Settings...` (*bindkey* **`Ctrl+Shift+E`**)

S'obre una finestra, on s'ha d'escollir la categoria **`Simulator Settings`**. Apareix el formulari que mostra la Fig. 4.14. En el camp **`Simulation mode`** heu d'escollir l'opció **`Timing`**.

Quan s'acabin de generar les formes d'ona per a la simulació amb retards i s'hagin d'emmagatzemar en un fitxer, hi ha dues opcions:

— se sobreescriu l'anterior fitxer d'ones i, per tant, el fitxer de *waveforms* manté el mateix nom
— es desa en un altre fitxer amb un altre nom i s'indica aquest nom al camp '**`Simulation input`**' de la finestra de la Fig. 4.14 (aquesta segona opció és la preferible). Per això, a l'exemple de la figura apareix el nom **ctrlSeq2.vwf**.

A continuació, creeu un nou fitxer de *waveforms* on establiu els vectors de simulació, executant l'ordre:

`File → New...`

S'obre un formulari on heu de seleccionar la pestanya **`Verification / Debugging Files`** i escollir l'opció **`Vector Waveform File`**. Finalment heu de prémer el botó **`OK`**. Apareix una finestra corresponent a l'editor de *waveforms* en la qual s'introdueixen els vectors de simulació (vegeu la Fig. 4.15).

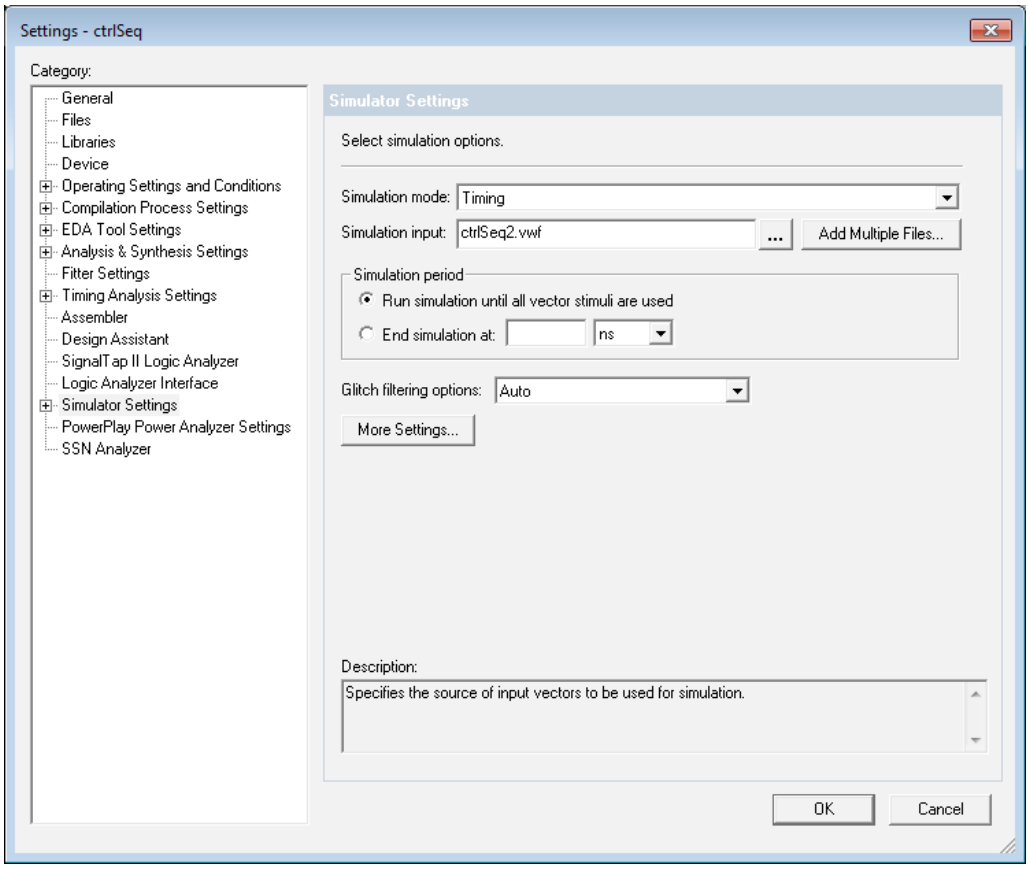


Fig. 4.14 Formulari amb les opcions de simulació (s'ha escollit fer una simulació amb retards)

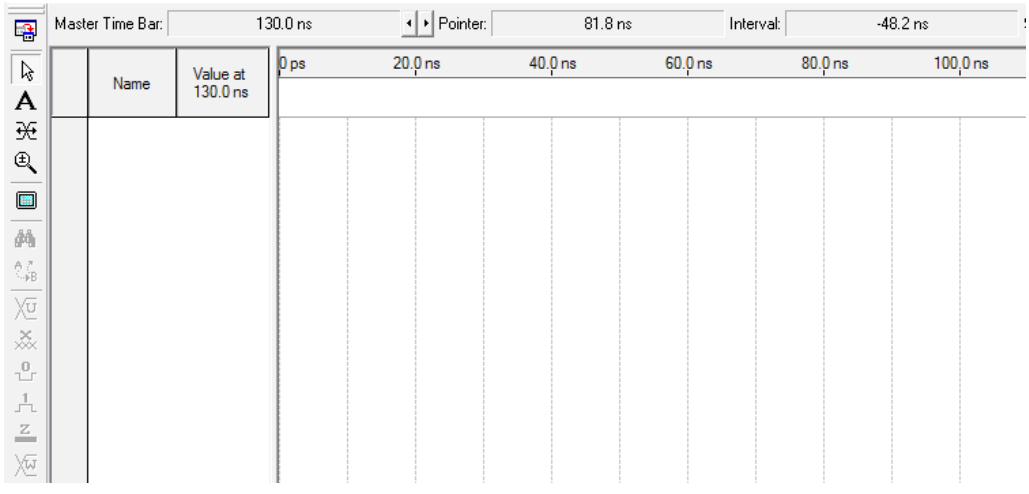


Fig. 4.15 Finestra d'edició de formes d'ona

Abans d'establir els vectors de simulació indiqueu el temps que se simularà amb l'ordre (vegeu la Fig. 4.16):

`Edit → End Time` (poseu 1μs)

També heu d'establir l'amplada de la graella (*grid*) d'edició de les ones. Aquesta graella després ens ajudarà a especificar les ones. Executeu l'ordre (vegeu la Fig. 4.17):

`Edit → Grid Size` (poseu 10ns)

4.3.2. Elecció dels senyals

El primer pas consisteix a indicar per a quins senyals s'estableixen els vectors de simulació. Per fer-ho, heu de situar el ratolí sobre el botó **`Name`** i prémer el botó dret del ratolí. Com a resultat apareix el submenú de la Fig. 4.18 on heu de fer un clic sobre l'opció **`Insert → Insert Node or Bus`**.

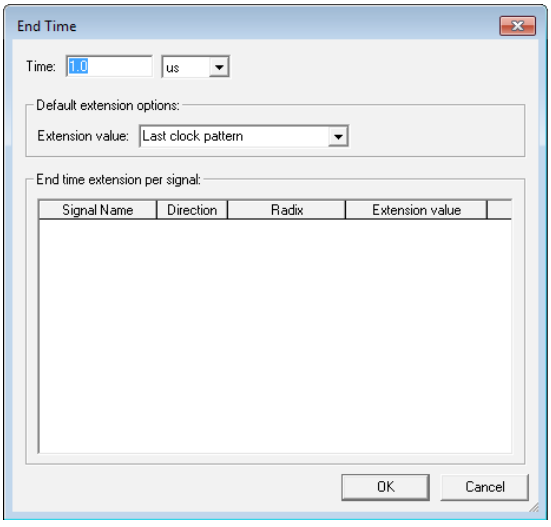


Fig. 4.16 Especificació de la duració de la simulació

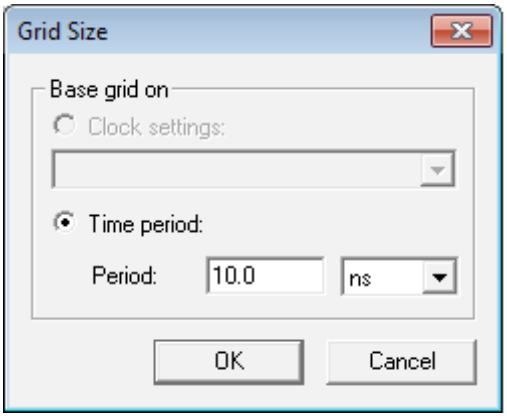


Fig. 4.17 Especificació de la grandària de la graella de l'editor de formes d'ona

Apareix una nova finestra (Fig. 4.19) en la qual es fa un clic sobre el botó **`Node Finder...`** . Automàticament s'obre la finestra mostrada a la Fig. 4.20, des de la qual se seleccionen els nodes d'entrada del circuit. La manera més senzilla de fer-ho és seleccionant a la finestra **`Filter`** l'opció **`Pins: Input`** i prement el botó **`List`** (vegeu les fletxes a la Fig. 4.20).

A la columna esquerra de la finestra apareix una llista amb totes les entrades que s'han posat a l'esquema. Per utilitzar-les s'han de passar a la columna de la dreta. Es pot fer de dues maneres: o se seleccionen les entrades que es necessiten i es prem la icona **>** (vegeu la fletxa 1 a la Fig. 4.21) o directament es prem la icona **>>** (assenyalada per la fletxa 2 a la Fig. 4.21), la qual passa tots els elements de la columna esquerra a la dreta. Després de seleccionar tots els senyals desitjats, premeu el botó **`OK`**.

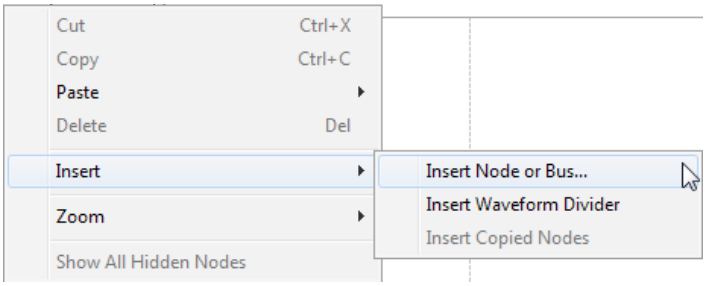


Fig. 4.18 Addició d'entrades a l'editor de formes d'ona (I)

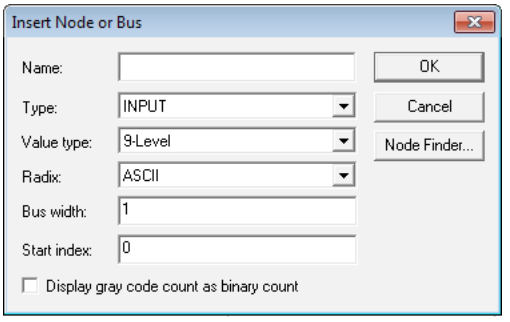


Fig. 4.19 Addició d'entrades a l'editor de formes d'ona (II)

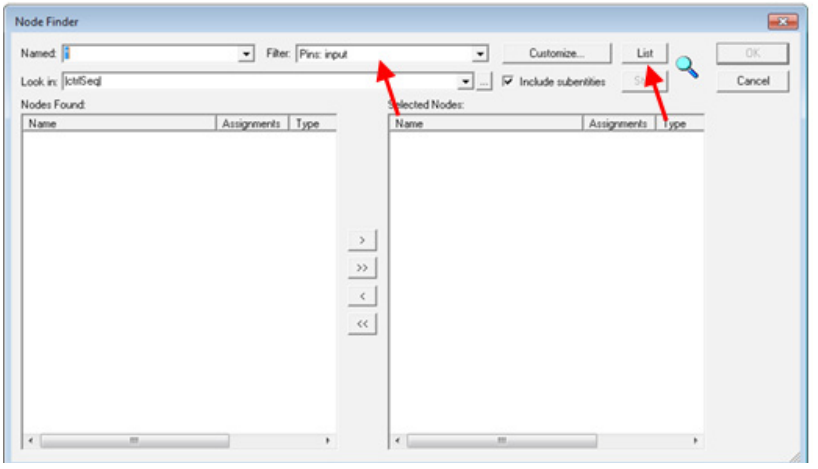


Fig. 4.20 Addició d'entrades a l'editor de formes d'ona (III

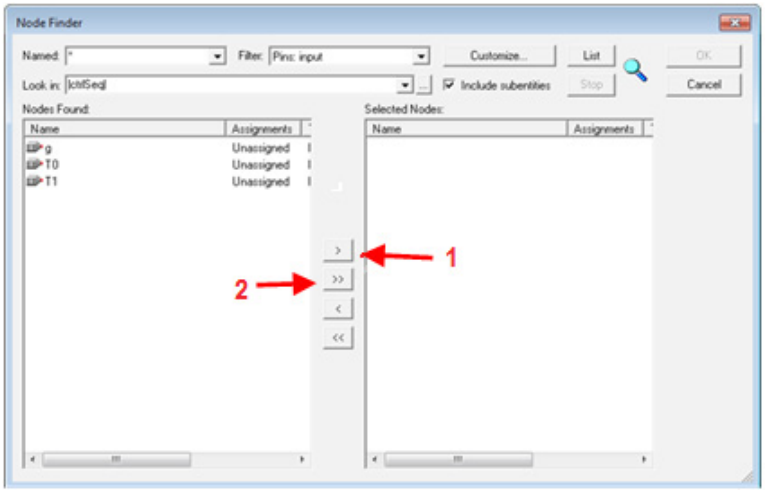


Fig. 4.21 Addició d'entrades a l'editor de formes d'ona (IV)

4.3.3. Especificació d'una forma d'ona

Arribats a aquest punt hem d'especificar la forma d'ona que volem aplicar a cadascuna de les entrades. Bàsicament hi ha quatre possibilitats. En cada simulació que fem durant el curs i per a cada entrada cal escollir la que ens interessi més. Les quatre possibilitats citades són les següents:

a) aplicar una sèrie de valors successius (0, 1, 2, 3, etc.) a un grup de senyals o bus (secció 4.3.3.1)
b) aplicar un valor constant a una entrada (secció 4.3.3.2)
c) aplicar una forma d'ona periòdica a un senyal d'entrada (secció 4.3.3.3)
d) aplicar una forma d'ona amb unes característiques concretes (secció 4.3.3.4)

En l'exemple que estem desenvolupant apliquem la primera possibilitat, que s'exposa a la secció següent.

4.3.3.1. Aplicació d'una sèrie de valors numèrics successius a un grup de senyals o bus

En aquest cas heu de definir un grup de senyals. No obstant això, abans de definir el grup, heu de posar en l'ordre desitjat els senyals que formen el grup. Per moure un senyal, l'heu de seleccionar i després, mantenint el botó esquerre polsat, arrossegar-lo fins a la posició desitjada. En funció de la posició que ocupin els senyals, aquests són considerats de més o menys pes.

Una vegada es troben en l'ordre correcte, seleccioneu tots els senyals i, prement el botó dret, executeu l'ordre **`Grouping`** → **`Group`** (Fig. 4.22). Apareix aleshores una finestra, on heu d'indicar:

`Group Name`: nom que es vol donar al grup
`Radix`: format en què es vol visualitzar el valor del grup

Finalment premeu el botó **`OK`** amb la qual cosa el grup ja queda definit.

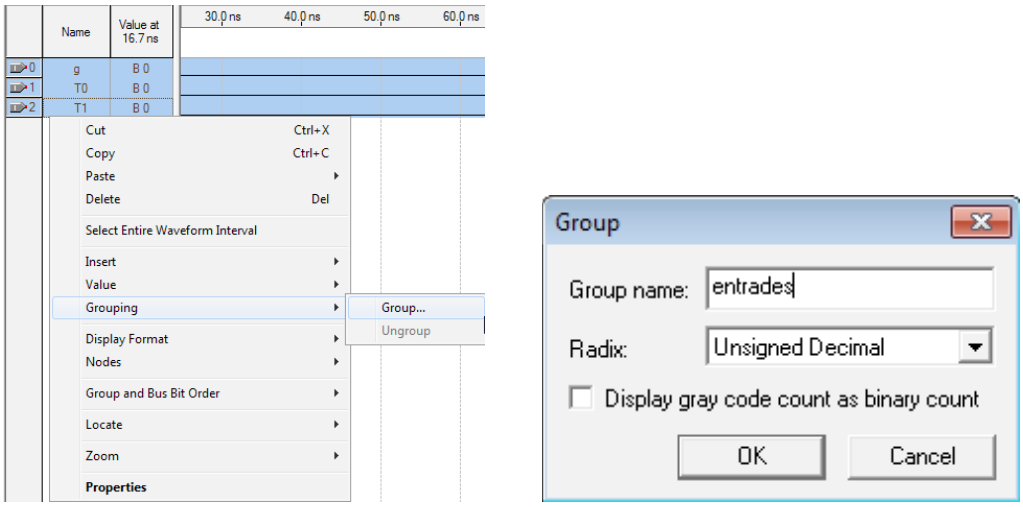


Fig. 4.22 Creació d'un grup de senyals

L'últim pas és definir el període del compte. Per fer-ho, heu de prémer la icona del comptador (Fig. 4.23). Automàticament s'obre la finestra de la Fig. 4.24 on heu d'especificar el valor dels camps de les pestanyes **`Counting`** i **`Timing`**:

Pestanya **`Counting`**:

`Radix`: arrel usada per visualitzar a la finestra el valor del grup d'ones. L'arrel també és utilitzada per interpretar l'edició del valor del grup d'ones.
`Start value`: valor inicial del compte
`End value`: valor final del compte
`Increment by`: valor de l'increment del compte

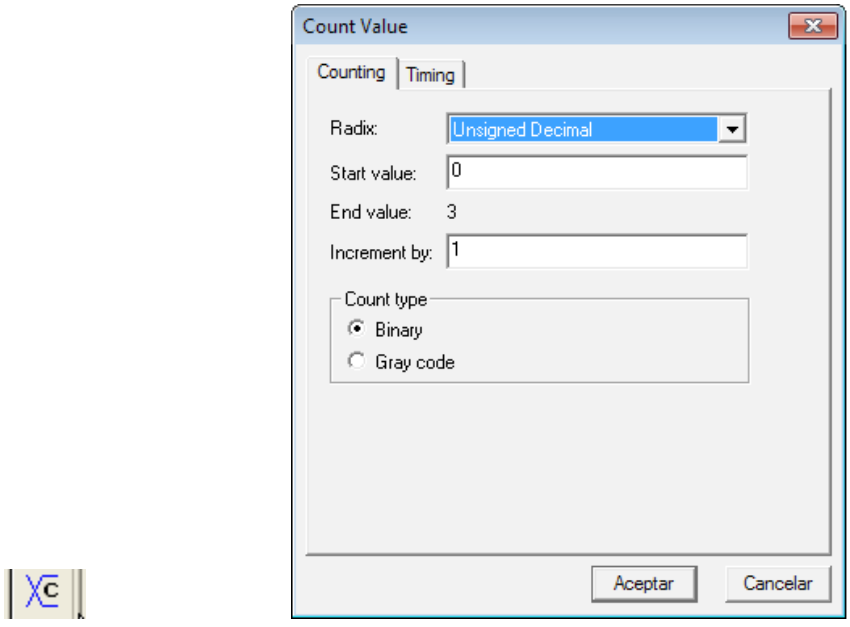


Fig. 4.23 Botó de l'ordre de definició d'un comptador
Fig. 4.24. Formulari associat a la definició d'un comptador

Pestanya **`Timing`**:

* **`At absolute times`**:
`Count every`: valor del període del compte (p.e., 50ns)

A l'hora d'escollir el període s'ha d'agafar prou gran, en funció del temps de retard del subcircuit el qual ataquen els senyals del grup.

Finalment premeu el botó **`Acceptar`** i podreu observar un contingut semblant al de l'exemple de la Fig. 4.25. Pitjant sobre les icones **+/–** podeu desplegar/plegar els senyals que conformen el grup.

Si en alguna ocasió us interessa destruir un grup de senyals, heu de seleccionar-lo, prémer el botó dret del ratolí i executar l'ordre **`Grouping → Ungroup`**.

IMPORTANT: Fins aquí l'explicació d'aquesta modalitat d'especificació d'ones. A continuació, a les seccions següents, s'exposen les altres tres modalitats. No les heu d'utilitzar per fer la simulació amb retards que esteu desenvolupant ara mateix, però us seran útils quan feu més simulacions en el futur. Per continuar fent la simulació actual podeu saltar a la secció 4.3.4 .

4.3.3.2. Aplicació d'un valor constant a una entrada

En primer lloc s'escull l'entrada i a continuació es prem el botó corresponent al valor constant que es vol establir. A la Fig. 4.26 podeu observar, de dalt a baix, els botons corresponents als valors '*no inicialitzat*', '*desconegut*', '*0*', '*1*' i '*alta impedància*'.

4.3.3.3. Aplicació d'una forma d'ona periòdica

Es comença seleccionant l'entrada i a continuació es prem la icona del rellotge (vegeu la Fig. 4.27). Aleshores apareix la finestra de la Fig. 4.28 on s'indiquen les característiques del senyal (període, fase, etc.). A l'hora d'escollir el període s'ha d'agafar prou gran tenint en compte el temps de retard del subcircuit el qual ataca el senyal. Aquesta manera d'especificar una forma d'ona ha estat àmpliament utilitzada durant l'exemple de simulació funcional desenvolupat a la secció 4.2 .

4.3.3.4. Especificació d'una forma d'ona qualsevol

Per especificar una forma d'ona amb unes característiques concretes, heu d'indicar en quins trams adopta cada valor (*0*, *1*, *Z*). Per fer-ho, seleccioneu amb el ratolí la porció d'ona que desitgeu fixar a un determinat valor (vegeu l'exemple de la Fig. 4.29). Aleshores prement amb el botó esquerre del ratolí la icona pertinent de la Fig. 4.30 establiu el valor que us interessa en aquell tram d'ona. Repetint aquest procés en diferents trams del senyal podeu obtenir una forma d'ona amb les característiques desitjades.

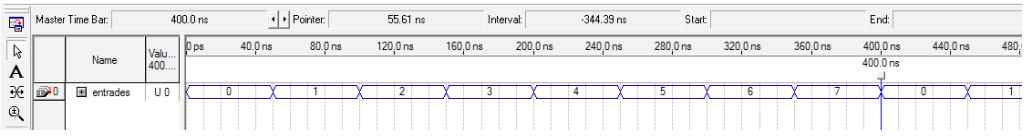


Fig. 4.25 Entrades agrupades amb comptador associat

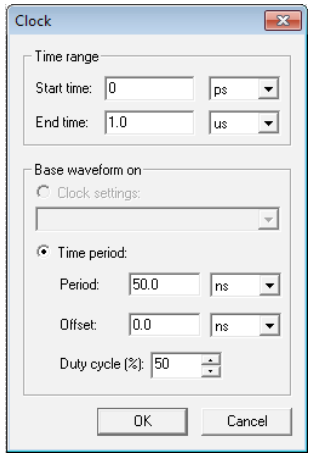


Fig. 4.26 Botons per a l'especificació de valors constants a les entrades
Fig. 4.27 Botó per a la generació d'un senyal periòdic
Fig. 4.28 Formulari associat a la generació d'un senyal periòdic

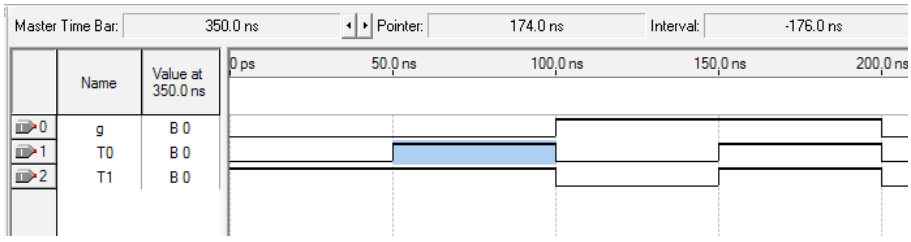


Fig. 4.29 Especificació d'una forma d'ona de manera lliure
Fig. 4.30 Botons per fixar el valor d'un tram d'ona

4.3.4. Gravació del fitxer d'ones. Especificació del fitxer d'entrada de simulació

Quan s'acabin de generar les formes d'ona es procedeix a emmagatzemar el fitxer amb l'ordre:

`File → Save As...`
especificant el nom desitjat del fitxer (en aquest exemple **ctrlSeq2**).

Per indicar al simulador que aquest és el fitxer amb els estímuls que s'han d'utilitzar a la simulació següent, s'executa l'ordre:

`Assignments → Settings` (*bindkey* **`Ctrl+Shift+E`**)

S'obre un formulari, on s'ha d'escollir l'entrada **`Simulator`**. Apareix la finestra que mostra la Fig. 4.31. S'emplena el camp següent:

`Simulation input`: Fitxer de formes d'ona que acabem de crear (en el nostre exemple **ctrlSeq2.vwf**).

Finalment es prem el botó **`OK`**.

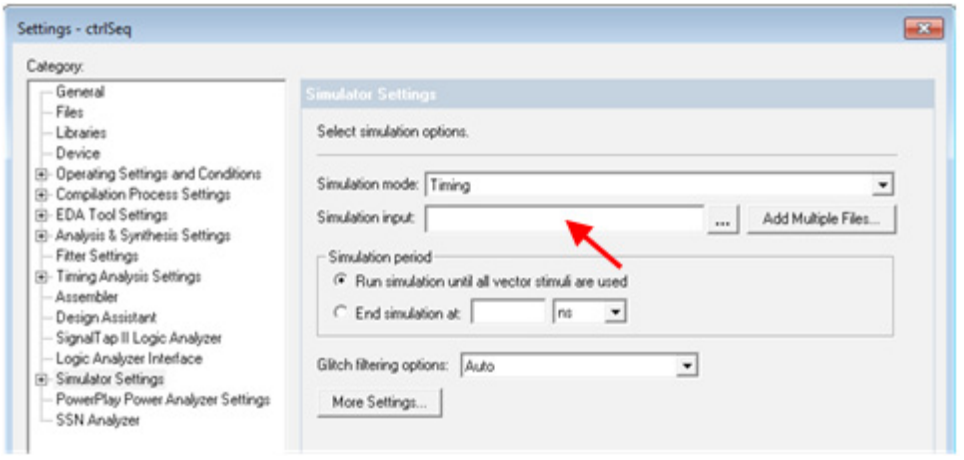


Fig. 4.31 Especificació del fitxer d'estímuls

4.3.5. Realització de la simulació

En aquest punt ja podeu realitzar la simulació. Per fer-la premeu el botó indicat per la fletxa de la Fig. 4.32. Una altra manera de llançar la simulació és executant l'ordre:

`Processing → Start Simulation` (*bindkey* **`Ctrl+I`**)

Al final de la simulació s'obre una nova finestra amb els resultats de la simulació com la que es mostra d'exemple a la Fig. 4.33, on podeu observar els retards als senyals de sortida (utilitzant la lupa).

Si es desitja gravar les formes d'ona de la finestra de sortida s'executa l'ordre:

`Save Current Report Section As...`
indicant el nom del fitxer, el qual habitualment té una doble extensió *.sim.vwf*.

Per obtenir una estadística dels temps de retard del circuit podeu executar l'ordre:

`Processing → Classic Timing Analyzer Tool → Pestanya tpd`

És interessant comprovar que els retards màxims són clarament inferiors al període dels comptadors definits per als grups de senyals. En cas contrari, la simulació pot presentar uns resultats no coherents. Si el període és massa petit, la simulació que s'obtindria no seria correcta, ja que estaríem fent treballar el mòdul dissenyat a una freqüència massa alta.

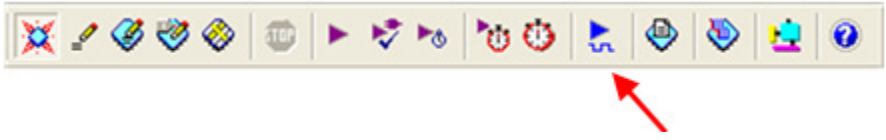


Fig. 4.32 Botó de simulació

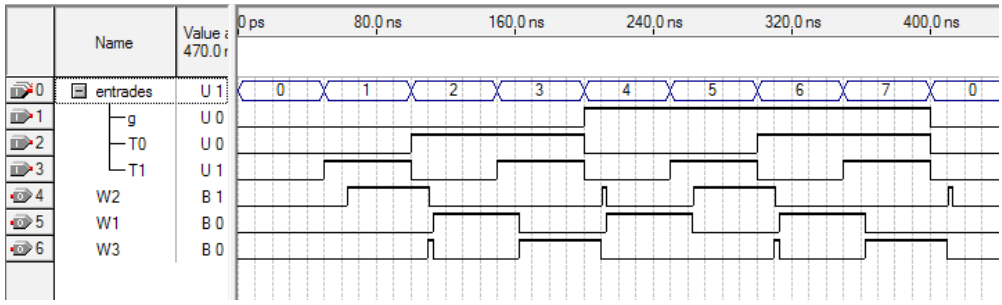


Fig. 4.33 Finestra amb els resultats de la simulació amb retards

4.4. Simulacions addicionals

Heu de fer dues simulacions amb retard addicionals:

- la primera d'elles és igual a l'anterior que heu fet amb la particularitat que el període de compte és tan sols de 10ns. Observeu cap diferència important en el resultat de la simulació en relació amb el cas anterior?

- la segona simulació consisteix a simular solament 4 vectors d'entrada (vegeu la Fig. 4.34) per a la qual cosa s'ha de seguir a la secció 4.3.3 el subapartat 4.3.3.4. Cada vector ha de tenir una durada de 50ns. Quin ha de ser el temps de durada de la simulació? Com s'estableix?

g	T0	T1	W1	W2	W3
0	0	1			
0	1	1			
1	0	0			
1	1	1			

Fig. 4.34 Senyals d'entrada per a la segona simulació addicional de ctrlSeq

5. Alguns mòduls bàsics de LittleProc

5.1. Introducció

A la part final d'aquest capítol es presenten les tres cel·les que s'han d'implementar amb l'editor d'esquemes de Quartus, durant la segona sessió de laboratori. Dues d'elles són mòduls bàsics de *LittleProc*. No obstant això, abans d'especificar les característiques d'aquests mòduls s'introdueix un seguit de coneixements relatius a certs aspectes de l'editor de Quartus que són utilitzats posteriorment.

5.2. Nom de cel·la i nom d'instància

Quartus té diverses biblioteques de cel·les que poden ser utilitzades per a la creació d'un circuit. Cada cel·la conté un nom que la identifica, l'anomenat "nom de la cel·la" (en anglès, *cell name*). Així, per exemple, la cel·la de tres entrades que constitueix l'element de memorització balda (en anglès, *Latch*), s'anomena LATCH. En un circuit es poden requerir diverses cel·les LATCH. A cadascuna de les repeticions d'una cel·la en un circuit se li diu instància (en anglès, *instance*). Cada instància té un nom que la identifica, l'anomenat "nom de la instància". Habitualment aquest nom és establert automàticament per l'editor de Quartus i adopta la forma **instx**, on **x** és un nombre. Tanmateix, el dissenyador pot canviar aquest nom i posar-ne un que consideri més oportú (sempre que no s'hagi usat en cap altre lloc del circuit). Hi ha un exemple il·lustratiu en un apartat posterior on es presenta un registre de baldes.

5.3. Baldes (*Latches*)

5.3.1. Funcionament d'una balda (o latch)

Una balda és un circuit electrònic utilitzat per emmagatzemar informació en sistemes lògics. Les baldes s'agrupen per formar registres. Les baldes són dispositius biestables (tenen 2 estats) que no posseeixen una entrada de rellotge i que canvien els estats de sortida només en resposta a dades d'entrada, en contraposició als *flip-flops* l'estat dels quals pot canviar solament en resposta a un flanc de rellotge o un senyal d'inicialització, ja sigui un PRESET o un RESET (*clear*).

A la Fig. 5.1 es pot apreciar una balda o *Latch*. Podem veure, com a entrades, l'entrada D de dades i l'entrada ENA d'habilitació, i el senyal de sortida Q.

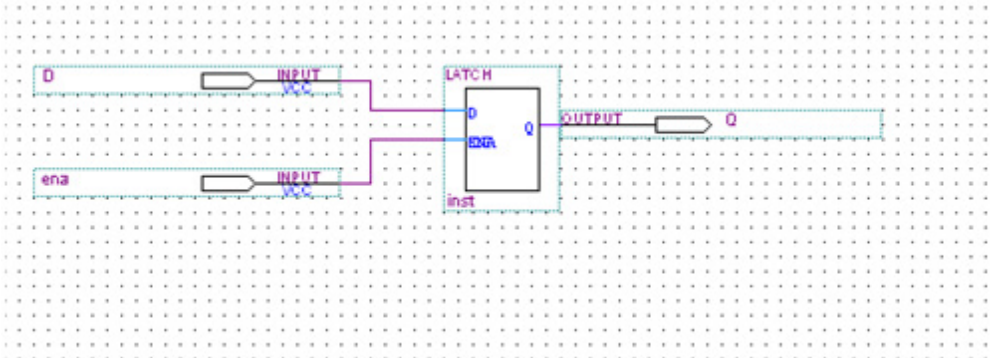


Fig. 5.1 Balda

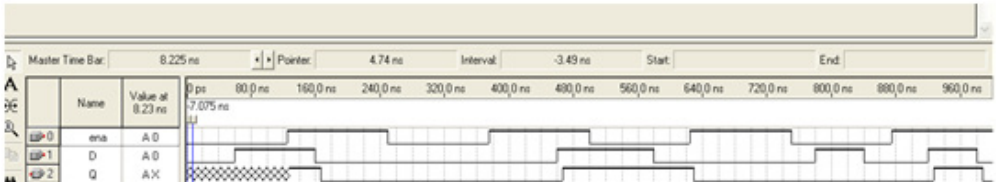


Fig. 5.2 Forma d'ones d'una balda

En la forma d'ona de la Fig. 5.2 podem veure que quan el senyal d'habilitació té el valor '1', tot el que hi ha a l'entrada D és el que es veu a la sortida Q (amb un retard de nanosegons a conseqüència de com està construït el dispositiu). Quan el senyal d'habilitació té el valor '0', la sortida Q no es modifica i a la balda queda emmagatzemat el que hi havia a D quan l'entrada ENA es va desactivar. En aquests moments qualsevol canvi a D no es veu reflectit a Q. El temps que la dada està estable a Q és el temps que ENA està desactivat.

5.4. Registre de baldes *(o latches)*

A la Fig. 5.3 podem veure un exemple del concepte d'instància. Es tracta d'un petit circuit format per 4 instàncies de la cel·la LATCH. La primera i l'última instància tenen com a noms **LSB** i **MSB**, i han estat posats pel dissenyador. La segona i la tercera tenen com a nom d'instància els noms **inst1** i **inst2** que, com hem dit abans, són els noms que l'eina posa per defecte.

Per modificar el nom d'una instància s'ha de fer clic dues vegades damunt de la instància i introduir el nom desitjat en el formulari que s'obre.

Aquesta cel·la és un registre de *latches*. En aquest cas el registre pot emmagatzemar 4 bits. És a dir, pot tenir valors que van des del D[0,1,2,3] = [0,0,0,0] fins al D[0,1,2,3] = [1,1,1,1].

Si feu el símbol d'aquest esquema (Fig. 5.4) fixeu-vos que teniu les 4 entrades D[i], l'entrada ENA d'habilitació i les 4 sortides Q[i]. Observeu que si en lloc d'haver-hi 4 entrades/sortides n'hi hagués 64, connectar els 64 cables es faria molt pesat, per això més endavant s'explicarà el concepte de línia de bus.

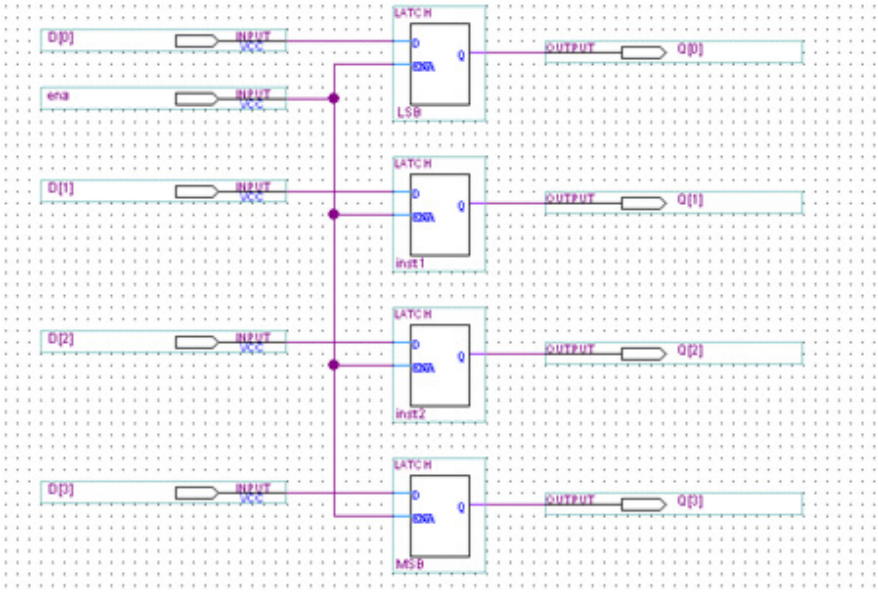


Fig. 5.3 Registre de 4 baldes

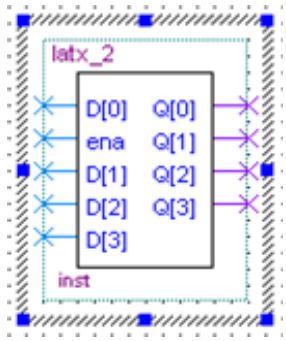


Fig. 5.4 Símbol del registre de 4 baldes

5.5. Flip-flops

5.5.1. Funcionament d'un flip-flop

A la Fig. 5.5 es mostra un *flip-flop*; com ja s'ha dit, aquest està controlat per un rellotge. El *flip-flop* que es mostra té les entrades prn, D, clk i clrn; i com a sortides té Q. El senyal de preset (prn) és l'entrada asíncrona per posar el *flip-flop* a 1 lògic i com que el terminal PRN del símbol DFF té una bola vol dir que és actiu a baixa. De la mateixa manera, clrn es l'entrada asíncrona de reset i també és activa a baixa.

A la forma d'ona de la Fig. 5.6 es veu que el senyal de rellotge clk té un període de 100ns. Fem un preset amb el senyal PRN al començament de la simulació (actiu a baixa) i després es desactiva durant tota la simulació. Fixeu-vos que quan hi ha un flanc de pujada del rellotge és quan el valor que hi ha a D s'emmagatzema a Q. El funcionament del *flip-flop* ve regit pel flanc de pujada del rellotge. La dada està estable a Q durant gairebé tot el període de rellotge menys el temps que hi ha el flanc.

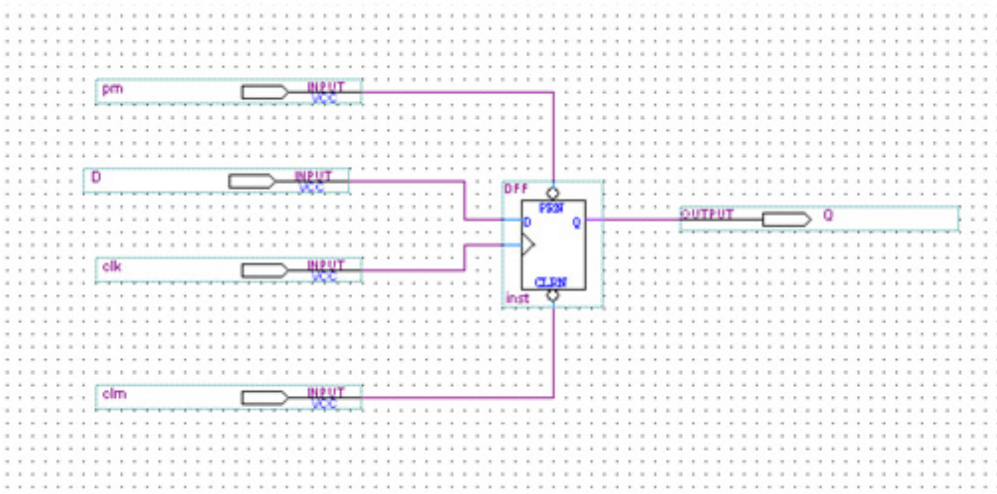


Fig. 5.5 *Flip-flop* D amb preset i reset asíncrons

Fig. 5.6 Formes d'ona d'un *flip-flop*

5.6. Connexió per nom

Per expressar la connexió entre els terminals de les portes d'un circuit s'utilitzen les línies de connexió. Ara bé, en un circuit amb moltes portes i connexions, si s'haguessin de dibuixar totes les línies de connexió s'obtindria un esquema molt complex i de molt baixa llegibilitat. Per evitar una situació com aquesta s'utilitza l'anomenada *connexió per nom*. Consisteix a donar nom a les línies de connexió. Aquest nom representa el nom del node elèctric al qual pertany la línia. Dues línies (o una línia i un pin) que tenen el mateix nom es considera que estan connectades elèctricament encara que al dibuix de l'esquema no es trobin connectades.

La Fig. 5.5 i la Fig. 5.7 són dos circuits equivalents. Si observeu les diferències entre les dues podeu veure que l'esquema de la Fig. 5.7 té totes les connexions per nom.

Per etiquetar (posar nom) a una línia, hi ha dues possibilitats. La primera (i més ràpida) consisteix, just després d'haver dibuixat la línia, a escriure'n el nom. La segona consisteix a seleccionar la línia i editar les seves propietats mitjançant l'ordre:

`Edit → Properties`

S'obre un formulari (vegeu la Fig. 5.8), en què s'indica el nom del node elèctric al qual està vinculada la línia.

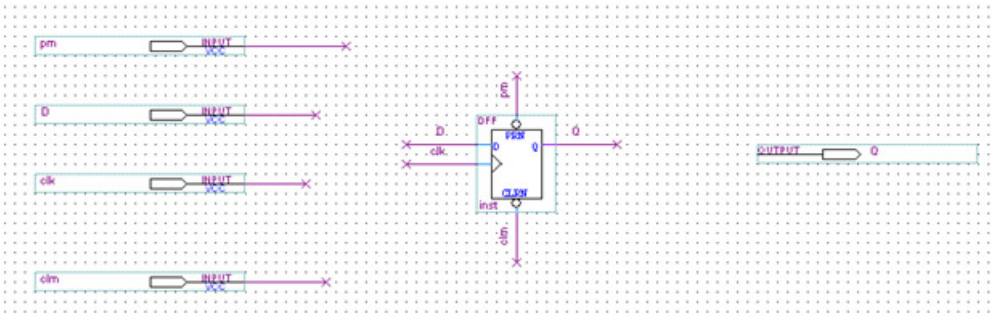


Fig. 5.7 Exemple de connexió per nom

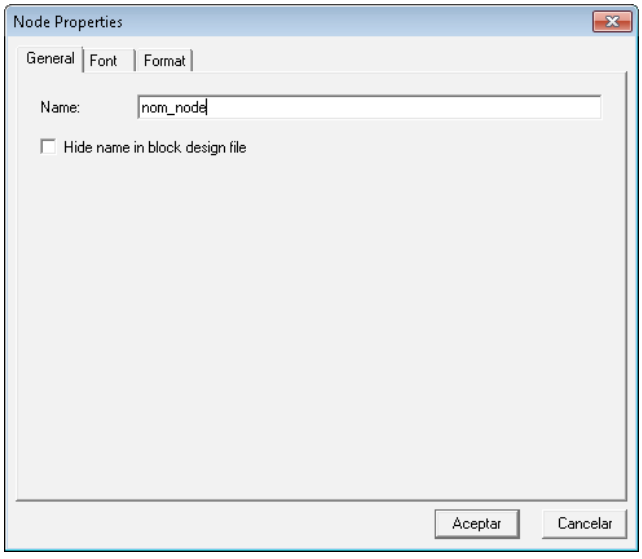


Fig. 5.8 Formulari per introduir el nom d'un node elèctric

ATENCIÓ

Un error molt comú quan s'etiqueta una línia és que no estiguem seleccionant la línia amb el ratolí de la manera apropiada; quan això succeeix el nom de la línia queda en verd i no en color grana com hauria de ser. A la Fig. 5.9 es mostra un etiquetat correcte i un altre d'incorrecte.

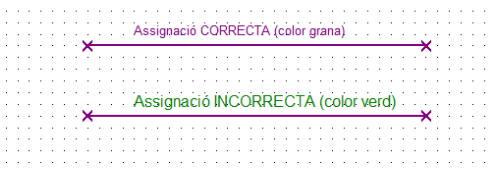


Fig. 5.9 Assignació de nom a una línia de maneres correcta i incorrecta

5.7. Especificació dels busos

Quan es vol especificar una línia de connexió d'un sol bit s'utilitza una línia fina ( ). En canvi, quan es vol especificar un bus de n bits s'utilitza una línia gruixuda ( ). Per dibuixar ambdós tipus de línies s'usen els botons i , respectivament.

Per expressar el nom d'un bus, s'indica el nom seguit d'una parella de claudàtors entre els quals s'indiquen els índexs dels bits. Per exemple, un bus de 4 bits que es vol que s'anomeni *D* s'especifica com a D[3..0]. Si es vol fer referència als dos bits centrals del bus, s'utilitza l'expressió D[2..1].

A la Fig. 5.10 es pot veure l'especificació d'un registre de 4 *flip-flops*, amb una entrada en aquest cas de tipus bus, és a dir, no entrades individuals D[0], D[1], D[2], D[3] sinó un bus D[3..0]. El mateix es pot dir amb la sortida, no sortides individuals Q[0], Q[1], Q[2],Q[3] sinó el bus Q[3..0].

Una altra possibilitat que ofereix Quartus és la creació d'un bus per agregació de senyals. Per expressar sintàcticament l'agregació s'utilitzen comes. Així, l'expressió *a,b,D[1..0]* denotaria un bus de 4 bits format pels senyals a, b i els dos últims bits del bus D.

5.8. Expressió compacta d'un conjunt d'instàncies

Hi ha situacions en les quals interessa utilitzar un conjunt d'instàncies on totes elles tenen una connectivitat molt similar. Uns casos ja coneguts són els dels registres de n *flip-flops* o de n baldes. Això es pot especificar a Quartus de dues maneres diferents. La primera, la manera clàssica, seria utilitzar n instàncies d'una cel·la DFF (o LATCH) on cadascuna d'elles tindria una entrada atacada per cadascun dels bits del bus (com ja s'havia vist a la Fig. 5.3 i també es pot observar a la Fig. 5.10). La segona, més còmoda i compacta, és utilitzar un sol símbol de la cel·la DFF (o LATCH), on una de les seves entrades és atacada pel bus de dades. En aquest cas, Quartus interpreta que aquest símbol representa n instàncies de la cel·la DFF (o LATCH), cadascuna d'elles atacada per un dels bits del bus.

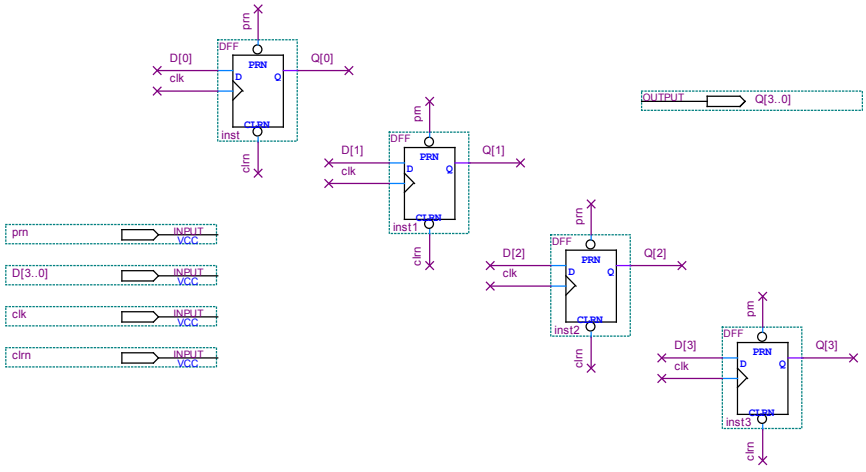


Fig. 5.10 Registre de *flip-flops* amb especificació de busos

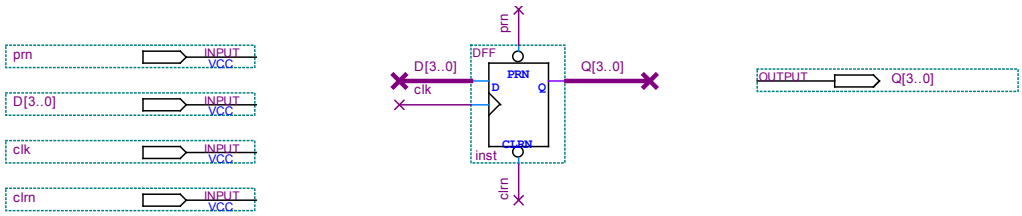


Fig. 5.11 Registre de *flip-flops* de 4 bits expressat de manera compacta

La Fig. 5.11 mostra un exemple del que hem comentat en el paràgraf anterior. El circuit d'aquesta figura és equivalent al de la Fig. 5.10. En ambdós casos és un registre de *flip-flops* de quatre bits. La versió de la Fig. 5.11 expressa el mateix però de manera compacta.

Observeu a la Fig. 5.11 que les entrades i sortides són com a la Fig. 5.10, però ara les línies connectades a l'entrada D i a la sortida Q de l'element instanciat (**DFF - inst**) són gruixudes. Això es deu a què representen sengles busos. Com es pot veure, ara l'entrada D i la sortida Q dels flip-flops en lloc de rebre 4 línies d'entrades i 4 línies de sortida reben una entrada i una sortida tipus bus amb els busos D[3..0] i Q[3..0].

Nota: Aquesta és la implementació més compacta i més eficient, i és d'aquesta manera, si no es diu el contrari, com heu de fer les vostres implementacions.

5.9. Cel·les a realitzar

5.9.1. Registre de baldes de 8 bits

Es demana que a partir de la informació d'aquest capítol es faci de manera compacta i se simuli un registre de baldes de 8 bits.

5.9.2. Registre de flip-flops de 8 bits (cel·la microPC)

La cel·la microPC és un registre de 8 bits que té les entrades i sortides següents.

Entrades

asynResetn: senyal de *reset* asíncron, actiu a baixa.
ck: senyal de rellotge amb flanc actiu de pujada.
datain[7..0]: bus de 8 bits amb les dades a carregar.

Sortides

dataout[7..0]: bus de 8 bits que indica el valor emmagatzemat al registre.

La funcionalitat del registre és molt simple. Quan s'activa el senyal **asynResetn**, el registre, de manera asíncrona, es posa a 0. Si el senyal **asynResetn** està inactiu, quan es produeix el flanc actiu de rellotge, el registre es carrega amb el valor que hi ha a l'entrada **datain**.

Per fer aquest registre s'han d'utilitzar *flip-flops d* (cel·la **DFF**) que trobareu a la secció *storage* de la biblioteca *primitives* (vegeu la Fig. 5.12).

Si feu un esquema clàssic del registre, utilitzant 8 flip-flops **DFF**, podreu observar que tots els *flip-flops* tenen una connectivitat molt semblant. Per aquest motiu se us

demana que feu un esquema compacte utilitzant un únic símbol de la cel·la **DFF**, aplicant els conceptes explicats a la secció 5.8 . Poseu *mPC* com a nom d'instància al símbol de la cel·la **DFF**. Amb aquest nom es fa referència al grup dels vuit *flip-flops*.

5.9.3. Cel·la PC

La cel·la PC és un registre de 12 bits que té les entrades i sortides següents.

Entrades

asynResetn: senyal de *reset* asíncron, actiu a baixa.
ck: senyal de rellotge; el flanc actiu és el de pujada.
init: senyal d'inicialització síncrona del registre a un valor predeterminat (valor binari 00000000xxxx), actiu a alta. 'xxxx' és el valor del vostre grup de treball. Per exemple, si el grup és el 7, el valor en binari seria 0111 i llavors la inicialització seria 000000000111. Si fos el grup 2, seria 000000000010, etc.
load: senyal de càrrega del registre, actiu a alta.
datain[11..0]: bus de 12 bits amb les dades a carregar.

Sortides

dataout[11..0]: bus de 12 bits que indica el valor emmagatzemat al registre.

La funcionalitat del registre es comenta a continuació. Quan s'activa el senyal **asynResetn**, el registre, de manera asíncrona, es posa a 0. Si **asynResetn** està inactiu, quan es produeix el flanc actiu de rellotge poden passar tres coses:

— si el senyal **load** no està actiu aleshores el registre manté el valor que tenia;
— si el senyal **load** està actiu, aleshores:
  — si el senyal **init** està actiu el registre es carrega amb el valor predeterminat 00000000xxxx;
  — si el senyal **init** no està actiu aleshores el registre es carrega amb el valor que hi ha a l'entrada **datain**.

Per fer aquest registre s'han d'utilitzar *flip-flops* D amb *enable*, també anomenats *flip-flops* E. La cel·la **DFFE** es correspon amb aquest tipus de *flip-flop* i la podeu observar a la Fig. 5.13. Podeu veure que té les mateixes entrades i sortides que un biestable de tipus D, juntament amb una entrada addicional, l'entrada ENA. Si l'entrada ENA val 1, quan es produeix el flanc actiu del rellotge el biestable carrega la dada que hi ha a l'entrada D; si, en canvi, l'entrada val 0, el biestable manté la informació que tenia (suposant que les entrades de *preset* i *reset* estiguin inactives).

Intenteu fer servir l'expressió compacta de conjunts d'instàncies. Els *flip-flops* de la cel·la es poden dividir en dos grups: els que durant la inicialització síncrona adopten el valor 1 i els que adopten el valor 0. Els *flip-flops* de cada grup presenten una connectivitat molt similar. Representeu cada grup amb un sol símbol de la cel·la **DFFE** i poseu *ff1* i *ff0* com els seus noms d'instància, respectivament.

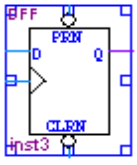


Fig. 5.12 Cel·la DFF

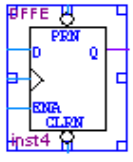


Fig. 5.13 Cel·la DFFE

6. VHDL llenguatge de descripció de maquinari

6.1. Introducció

Els llenguatges de descripció de *hardware* (HDL *Hardware Description Language*, per les seves sigles en anglès) proporcionen mitjans eficaços per al disseny assistit per computador (CAD) de xarxes lògiques digitals. VHDL (la "V" denota VHSIC, sigles utilitzades pel Departament de Defensa dels Estats Units per una àrea coneguda com a *Very High-Speed Integrated Circuits*) és només un llenguatge d'alt nivell que permet descriure totes les característiques importants d'una xarxa lògica de complexitat arbitrària. Incloses en els llistats apareixen funcions individuals, propietats dels senyals i detalls de com estan connectats els blocs per formar un sistema complet. Un cop el sistema està caracteritzat amb el format i la sintaxi apropiada, es compila en un programa HDL. El resultat és un conjunt de sortides que es poden fer servir per simular i verificar el comportament de la xarxa lògica.

Per tal de realitzar segons quines parts com la cel·la *ctrlseq, microPC* o la cel·la *PC* és millor (més didàctic) el desenvolupament amb captura d'esquemes. No obstant això, qualsevol cel·la de les que hem desenvolupat es podria descriure amb un llenguatge de descripció de maquinari com és el llenguatge VHDL.

Abans de llegir de manera detinguda tot aquest capítol es recomana que es llegeixi quins són els mòduls que s'han de realitzar; això ho trobareu al final del capítol. Després comenceu a llegir i tingueu en compte quines són les parts del capítol que us interessen per realitzar els quatre mòduls que s'han de fer en VHDL.

La manera de descriure un mòdul en llenguatge de descripció de maquinari (VHDL) amb l'eina de desenvolupament Quartus II és molt similar a l'entrada d'esquemes. S'ha de crear un nou fitxer: *FILE→New→Design Files* i s'ha de triar l'opció *VHDL file* (vegeu la Fig. 6.1). Llavors s'obre un fitxer d'escriptura amb l'extensió *vhd*. En aquest fitxer podeu agafar per exemple l'*adder.vhd* que està explicat una mica més avall i fer una síntesi, simulació, etc., per veure la similitud amb l'entrada d'esquemes (vegeu la Fig. 6.2).

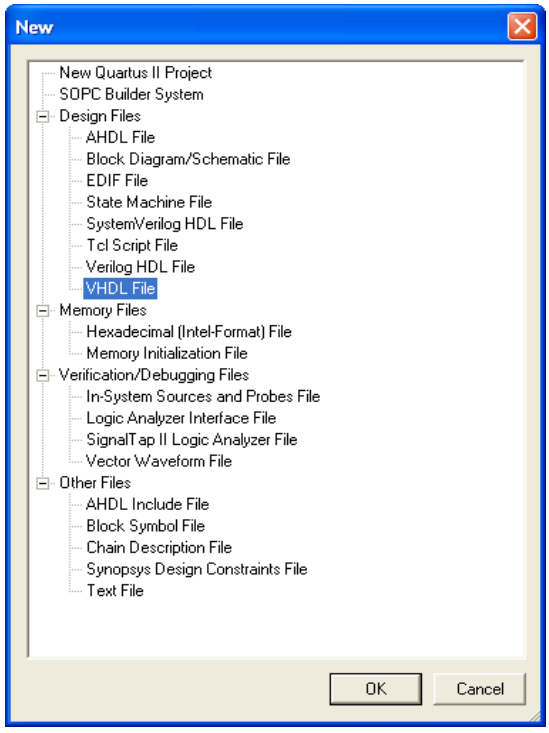


Fig. 6.1 Creació del fitxer VHDL

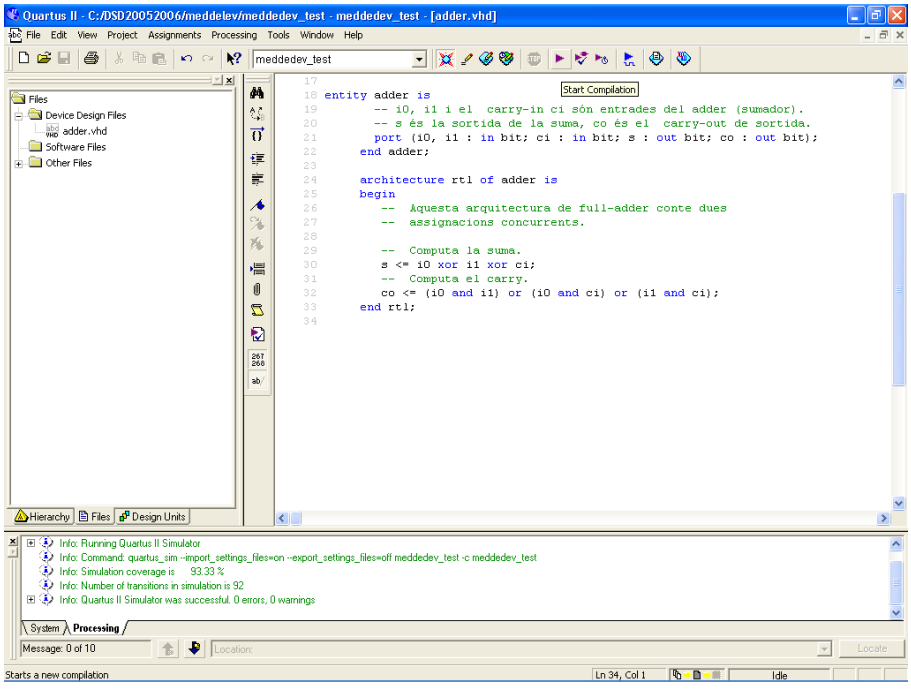


Fig. 6.2 Entrada escrita en VHDL de l´exemple adder.vhd

6.2. El llenguatge VHDL

En aquest apartat es pretén fer una petita introducció al llenguatge que s'utilitza per implementar certes parts del nostre processador elemental. En el disseny jeràrquic, qualsevol sistema es pot veure com fet de blocs més petits. Cada bloc proporciona una funció especifica i, al seu torn, pot estar format per blocs encara més petits. Si es comença amb les funcions més simples i després es construeixen blocs més complexos, estem donant un enfocament del disseny lògic conegut com a enfocament ascendent. A la inversa, es poden iniciar en el nivell més alt (el sistema) i dividir-lo en blocs lògics cada cop més petits, i això es diu enfocament descendent.

6.2.1. Conceptes bàsics

Es considera un bloc de funció lògica general com el que es mostra a la Fig. 6.3. La unitat es caracteritza per ports d'entrada que accepten variables lògiques A, B, C i un port de sortida que dóna la funció f(A,B,C). Es pot utilitzar aquest tipus de descripció sense importar la complexitat interna del bloc. No hi fa res si la funció lògica és complexa o un parell de portes lògiques.

Ara, suposem que volem descriure l'operació d'un bloc a algú amb qui estem parlant per telèfon. Els diagrames lògics requereixen la transmissió d'una imatge gràfica, o intentar descriure un diagrama amb paraules. Si bé la descripció verbal d'un circuit lògic pot ser adequada si la unitat és petita, no és pràctica per descriure una xarxa complexa; a més, hi ha una alta possibilitat d'error.

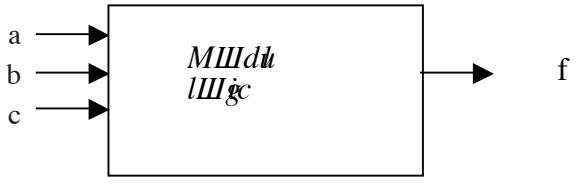


Fig. 6.3 Diagrama d'un bloc lògic

Hi ha un altre problema fins i tot si es té disponibilitat del diagrama lògic. Com provar el disseny lògic per assegurar que funcioni com es creu que ho hauria de fer? Un bon dissenyador sempre realitza molts nivells de prova per evitar canvis costosos si es troba un error a les etapes posteriors del cicle de disseny. Encara que pot realitzar una verificació lògica en rastrejar tota seqüència possible a través d'un diagrama lògic, aquest procediment resulta ser molt tediós i també introdueix un gran marge d'error.

El llenguatge de descripció de maquinari (hardware) proporciona la solució per als dos tipus de problemes. Permet descriure l'operació d'una xarxa digital en utilitzar enunciats definits amb precisió, en una forma que s'assembla a la d'un llenguatge de computadora d'alt nivell genèric tal com C++. A la inversa, és possible construir el diagrama lògic directament de la descripció HDL; això no obstant, els HDL tenen una aplicació més important, que és la capacitat de simular la xarxa lògica. La prova i el redisseny es poden realitzar a nivell del HDL, i qualsevol canvi o correcció es pot integrar al disseny original. Tal com implica el seu nom, un HDL permet incloure característiques del maquinari a la simulació. També pot afegir problemes físics com els retards de propagació per

les portes lògiques, per assegurar que el circuit funcionarà com desitgi el dissenyador. D'altra banda, permet trobar punts problemàtics i millorar el disseny.

Un altre aspecte útil del VHDL és que es pot utilitzar per descriure qualsevol xarxa digital. Tanmateix, es troba estandarditzat i s'utilitza en una gran varietat de plataformes.

6.2.2. Utilització del llenguatge de descripció de maquinari

Per usar un llenguatge de descripció de hardware, primer cal crear una descripció de la xarxa utilitzant la sintaxi i les construccions definides dins del mateix llenguatge. La descripció s'introdueix dins un arxiu de text, el qual es desa per a la seva utilització posterior. La Fig. 6.4 il·lustra aquest procés. El compilador de HDL accepta l'arxiu d'entrada i genera un arxiu de sortida que té tota la informació sobre la xarxa. És a dir, un arxiu amb informació tecnològica de com està implementat el nostre disseny.

Després, aquest arxiu es fa servir com a ambient de simulació amb un altre programa, el qual permet que l'usuari especifiqui els senyals d'entrada i vegi els resultats corresponents. Això es fa generalment a través d'una interfície gràfica que mostra els senyals a mesura que aquests canvien amb el temps.

Fig. 6.4 Ús d'un compilador VHDL

6.2.3. Definició de mòduls en VHDL

Una unitat lògica com la mostrada a la Fig. 6.3 es diu mòdul o unitat. Per descriure en VHDL un mòdul és necessari fer dues coses. Primer, cal definir el bloc donant-li un nom i especificant les línies d'entrada i de sortida (les quals es denominen ports). Això es defineix com a entitat (*entity*). Un cop s'introdueix l'entitat, cal especificar el que el mòdul fa, és a dir, la manera en la qual les sortides es relacionen amb les entrades, cosa que s'anomena arquitectura (*architecture*) del mòdul. L'arquitectura es pot descriure de maneres diferents, i això depèn del nivell de modelat desitjat.

Igual que qualsevol llenguatge de computadors, VHDL requereix que l'usuari tingui en compte una sèrie de regles que defineixen les paraules clau i la sintaxi. Una "paraula clau", també denominada paraula reservada, té un significat especial en el llenguatge, i no es pot utilitzar per a cap altre propòsit.

La "sintaxi" es refereix a l'ús de les paraules i a l'ordre en què s'han d'utilitzar per escriure enunciats, així com els requeriments de puntuació per definir punts especials en els enunciats, com el final d'una ordre. Les paraules reservades, així com la sintaxi, són detalls formals, i una part integral i necessària de VHDL, encara que el llenguatge proporciona un enfocament molt poderós a la simulació digital i la verificació que trans-

cendeix els detalls del codi escrit. Igual que amb qualsevol llenguatge de computadors, la familiaritat arriba amb l'ús. Per tant, la pràctica és molt recomanable, i s'anima el lector que apliqui les tècniques explicades en aquest llibre i d'altres que estan al final d'aquest document a les referències. En aquest llibre només s'explica el VHDL que és indispensable per fer el desenvolupament del microprocessador.

6.2.4. Unitats de disseny en VHDL

Hi ha quatre unitats o elements bàsics en VHDL que es fan servir per a la descripció d'un circuit. Aquests quatre elements són:

— les entitats
— les arquitectures
— les configuracions
— els paquets

En els apartats següents s'explica què és cadascun d'aquests elements i com són utilitzats.

6.2.4.1. Entitat

Una entitat serveix per definir la visió externa d'un circuit que aquest circuit presenta, és a dir, la interfície amb el seu entorn. VHDL separa aquesta visió externa de la implementació concreta del dispositiu per donar la possibilitat que aquesta quedi oculta. D'aquesta manera, després d'analitzar la declaració d'una entitat i, per tant, haver-la emmagatzemat en una biblioteca, aquesta entitat pot ser utilitzada en altres dissenys que només requereixin la interfície per ser utilitzats.

La sintaxi en VHDL per a la declaració d'una entitat és la següent:

```
entity identificador is
  [genèrics]
  [ports]
  [declaracions]
[begin sentències ]
end [entity][identificador];
```

L'identificador és el nom de l'entitat i serveix per poder referenciar-la més tard. En cas que es repeteixi l'identificador al final de la declaració, aquest ha de ser idèntic al definit a la primera línia o es detecta un error sintàctic durant l'anàlisi de la declaració.

Excepte la primera i l'última línia de la declaració, totes les altres són opcionals. Per tant, l'entitat mínima seria la següent:

```
entity identificador is
end;
```

Entre les parts opcionals de la declaració d'una entitat s'ha de destacar la declaració dels ports, que determinen la interfície del dispositiu amb l'exterior. Per comprendre què són els ports d'una entitat es pot fer una comparació d'aquests i els terminals d'un símbol.

Per a cada port s'ha d'indicar el nom, el tipus i el mode. El nom s'utilitza per referenciar-lo, el tipus defineix la classe d'informació que es transmet pels ports, mentre que els ports poden ser d'entrada, de sortida o bidireccionals. Per exemple, la declaració d'una entitat que implementa un multiplexor de 2 bits seria:

```
entity MUX is
  port (a : in bit;
        b : in bit;
        sel : in bit;
        f : out bit);
end MUX;
```

A la Fig. 6.5 es veu un diagrama equivalent. Com s'hi pot apreciar, només s'indica com s'ha de comunicar el dispositiu amb el seu entorn sense donar cap detall d'implementació.

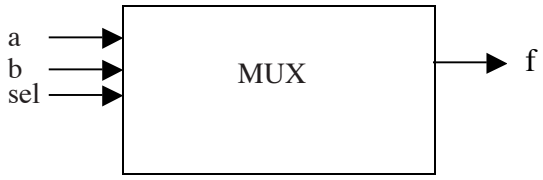


Fig. 6.5 Diagrama d'un MUX de 2 entrades

6.2.4.2. Arquitectura

L'arquitectura és una unitat del llenguatge en què es descriu un mòdul determinat (dispositiu), prèviament declarat mitjançant una entitat. La descripció es fa mitjançant les anomenades sentències concurrents que descriuen accions que es produeixen simultàniament. Existeixen diversos tipus de sentències concurrents. Una d'elles, la sentència *process*, serà explicada més endavant. La descripció de la funcionalitat es pot fer utilitzant tres estils descriptius diferents:

— un estil comportamental (s'especifica com opera el mòdul);
— un estil estructural (s'especifica les cel·les/subblocs que conformen el mòdul i com estan interconnectats);
— un estil híbrid on es barregen els dos estils anteriors.

Aquesta unitat del llenguatge s'expressa mitjançant la sentència *architecture*:

```
architecture identificador of entity_ident is
  [declaracions]
begin
....
  processos i/o altres sentències concurrents
....
end [architecture] [identificador];
```

Tota arquitectura ha d'estar vinculada a una entitat. Una arquitectura no té sentit per si sola. El que sí que és possible és que per a una entitat hi hagi diverses arquitectures definides (per exemple, amb diferents nivells d'abstracció o diferents estils descriptius).

Processos

Una sentència concurrent s'utilitza per descriure el comportament d'un mòdul d'un circuit (o d'una part d'un mòdul). Un procés és un tipus de sentència concurrent que defineix el seu comportament mitjançant un conjunt de sentències seqüencials. La sintaxi general d'un procés és:

```
[etiqueta:] process [(llista sensibilitat)] [is]
  declaracions
begin
  sentències seqüencials;
end process [etiqueta];
```

L'etiqueta del procés és opcional i serveix per identificar-lo. La partícula *is* té també caràcter opcional (habitualment no la inclourem). La llista de sensibilitat està constituïda pels senyals, el canvi dels quals pot provocar canvis en les sortides o en l'estat del mòdul (o submòdul) descrit. En VHDL és possible definir processos que no tinguin llista de sensibilitat. No obstant això, tots els exemples de processos que es presenten en aquest llibre i tots els processos que haureu de desenvolupar en els exercicis són processos amb llista de sensibilitat. Les variables (amb els seus valors inicials si els tenen) i les constants locals del procés s'especifiquen en l'apartat de declaracions. El comportament del procés (i del mòdul descrit) es defineix a través d'un algorisme expressat mitjançant una sèrie de sentències seqüencials.

Un procés, per defecte, es un bucle infinit que, en arribar al seu final, torna a la seva primera sentència seqüencial i continua la seva execució. No obstant, si el procés té una llista de sensibilitat, el procés en arribar a la seva última sentència s'atura fins que algun dels senyals que pertanyen a la llista de sensibilitat experimenta un esdeveniment (un canvi en el seu valor), la qual cosa provoca l'activació de nou del procés, és a dir, l'execució des de l'inici de les sentències seqüencials.

Com a norma general, quan s'utilitzi un procés per descriure el comportament d'un mòdul combinacional, la llista de sensibilitat estarà formada per totes les entrades del mòdul (un canvi a qualsevol entrada pot provocar canvis a les sortides). En el cas de mòduls seqüencials, la llista de sensibilitat inclourà els senyals de rellotge i el *preset*/*reset* asíncron, ja que quan varien aquests senyals el mòdul seqüencial pot canviar el seu estat o les seves sortides.

6.2.4.3. Configuracions

La configuració és la construcció VHDL encarregada de seleccionar l'arquitectura que es vol utilitzar per a una entitat concreta. Evidentment això és útil només quan hi ha diverses arquitectures per a una mateixa entitat. En principi, vosaltres només heu de desenvolupar una arquitectura per a cada entitat, per la qual cosa no heu de fer servir les configuracions.

6.2.4.4. Paquets

Un paquet permet agrupar un conjunt de declaracions perquè puguin ser utilitzades per diversos dispositius sense ser repetides en la definició de cada dispositiu. D'aquesta manera es facilita la reutilització i l'actualització del codi. La sintaxi VDHL per declarar un paquet és la següent:

```
package identificador
  [declaracions]
end [package][identificador];
```

La sintaxi en VHDL per al cos del paquet és:

```
package body identificador is
  [cos de les declaracions]
end [package body][identificador];
```

VHDL té dos paquets incorporats per defecte: Standard i Textio.

6.2.5. Biblioteques

Una biblioteca conté un conjunt de paquets. Les biblioteques són útils perquè faciliten la compartició i la reutilització del codi en diferents dissenys. Per donar visibilitat a una biblioteca s'utilitza la sentència *library* seguida del nom de la biblioteca. Per exemple, per utilitzar els elements d'un paquet que es digui *PaquetExemple* emmagatzemat a la biblioteca *BibliotecaExemple* des d'un model que s'hagi de desar en una altra biblioteca s'hauria de començar el model d'aquesta manera:

```
library BibliotecaExemple;
use BibliotecaExemple.PaquetExemple.all;
```

Les biblioteques *work* i *std* són excepcions en el sentit que sempre són visibles i, per tant, no requereixen la sentència *library*.

El paquet *std_logic_1164* acostuma a estar emmagatzemat en una biblioteca anomenada IEEE i no forma part del llenguatge VHDL. Per aquesta raó en cap model no pot ser utilitzat el tipus de dada *std_logic* directament. Els models que vulguin utilitzar aquest tipus de dada han de contenir la sentència necessària per obtenir la visibilitat d'aquest paquet. Concretament a l'inici s'ha d'incloure:

```
library IEEE;
use IEEE.std_logic_1164.all;
```

6.2.6. Variables i senyals

6.2.6.1.Variables

Habitualment, una variable s'utilitza per emmagatzemar valors intermedis dins un procés. Es declara en la part declarativa dels processos i solament és visible dins el procés on ha estat declarada. La sintaxi VHDL per declarar una variable és la següent:

```
variable identificador {,...} : tipus [:= expressió];
```

Alguns exemples:

```
variable index1, index2 : integer := 0;
variable comparació : boolean;
variable resultat : real;
```

Una variable pot canviar el seu valor mitjançant la sentència d'assignació de variables. Aquest canvi es produeix de manera immediata. L'operador d'assignació de variables és :=. Aquest operador també s'utilitza per inicialitzar les variables.

6.2.6.2. Senyals

Els senyals es fan servir per transmetre informació entre processos (o, dit de manera més general, entre sentències concurrents). Els senyals intervenen en la sincronització de l'execució i suspensió de processos. Es declaren en la part declarativa de la sentència *architecture* i són visibles en qualsevol procés (o sentència concurrent) definit dins l'*architecture*. La sintaxi VHDL per declarar un senyal és molt semblant a la sintaxi per declarar variables.

```
signal identificador {,...} : tipus [:= expressió];
```

A continuació se'n mostren alguns exemples:

```
signal rellotge : std_logic := 0;
signal comparació : bit;
signal resultat : integer range 0 to 7;
```

Un senyal és un objecte que, com una variable, pot canviar el seu valor mitjançant la sentència d'assignació de senyals. A diferència de les variables, l'assignació de senyals no es realitza de manera immediata sino que abans han d'acabar-se d'executar els processos actius. L'operador d'assignació de senyals és <=. Per inicialitzar un senyal s'utilitza l'operador := . Al primer exemple es pot veure com s'inicialitza el senyal *rellotge*.

Tots els *ports* definits a les entitats tenen estatus de senyals.

6.2.6.3. Tipus de dades. Operadors

VHDL compta amb els següents tipus de dades, definits en el paquet *Standard*:

bit	(0,1)
bit_vector	(array de bits, busos)
boolean	(true, false)
integer	(enters de 32 bits. Possibilitat de definir subrangs)
natural	$(0 – 2^{32} -1)$
positive	$(1 – 2^{32} -1)$
character	(caràcters ASCII)
string	(cadenes de caràcters)
time	(números que expressen temps, per exemple 20ns)

Aquests tipus poden ser utilitzats quan es declaren variables o senyals.

Les expressions en VHDL han de ser construïdes utilitzant els operadors mostrats a la taula 6.1 que està disposada en ordre de precedència. La precedència desitjada pot ser també aconseguida a través d'una utilització explícita dels parèntesis.

Operador	***Funció***	***Operand Tipus***
`and`	a & b	bit, bit_vector, boolean
`or`	$a \mid b$	bit, bit_vector, boolean
`nand`	!(a & b)	bit, bit_vector, boolean
`nor`	!($a \mid b$)	bit, bit_vector, boolean
`xor`	$a \wedge b$	bit, bit_vector, boolean
`=`	$a = b$	bit, bit_vector, boolean
`/=`	$a \neq b$	bit, bit_vector, boolean
`<`	$a < b$	bit, bit_vector, boolean
`<=`	$a \leq b$	bit, bit_vector, boolean
`>`	$a > b$	bit, bit_vector, boolean
`>=`	$a \geq b$	bit, bit_vector, boolean
`+`	$a + b$	integer (enter), real
`-`	$a - b$	integer (enter), real
`&`	*a concatenació b*	bit, bit_vector, character, string
`+`	$+ a$	integer (enter), real
`-`	$- a$	integer (enter), real
`*`	$a * b$	integer (enter), real
`/`	a / b	integer (enter), real
`mod`	a mòdul b	integer (enter)
`rem`	a residu b	integer (enter)
`**`	a^b	integer (enter), real
`abs`	$\mid a \mid$	integer (enter), real
`not`	$\neg a$	bit, bit_vector, boolean

Taula 6.1 Operadors

Per la seva banda, el paquet std_logic_1164, NO incorporat per defecte en VHDL, inclou els tipus de dades següents:

std_logic
std_logic_vector
std_ulogic
std_ulogic_vector

D'interès especial resulten els tipus *std_logic* i *std_logic_vector*. El tipus *std_logic* està format pels valors següents:

'0' : 0 fort
'1' : 1 fort
'X' : desconegut fort
'L' : 0 dèbil
'H' : 1 dèbil
'W': desconegut dèbil
'Z' : alta impedància
'-' : no importa (don't care)
'U' : no inicialitzat

Tots aquests valors representen diversos estats en què es poden trobar els nodes d'un circuit.

Per la seva banda, el tipus *std_logic_vecto*r fa referència a un *array* de dades de tipus *std_logic*. Per tant, pot ser utilitzat quan es vol treballar amb busos. Es recomana utilitzar aquests tipus en lloc dels tipus *bit* i *bit_vector*. És important recordar que per poder-los usar en el codi VHDL s'ha d'incloure el paquet *std_logic_1164*:

```
library IEEE;
use IEEE.std_logic_1164.all;
```

6.2.7. Atributs

Un atribut és una característica (una informació) que es pot associar a qualsevol element (per exemple, un senyal o un tipus de dades) d'un model fet amb VHDL. VHDL incorpora un conjunt d'atributs predefinits que poden ser utilitzats directament per l'usuari quan desenvolupa un model. La sintaxi VHDL per fer servir un atribut d'un element és:

```
nomElement'nomAtribut
```

Un exemple d'atribut predefinit en VHDL és l'atribut *event* que es de tipus booleà i és aplicable sobre senyals. Així, si S és un senyal, l'expressió *S'event* retorna el valor *true* si el senyal S ha experimentat un esdeveniment (un canvi de valor) i *false* en cas contrari.

VHDL també permet que l'usuari defineixi nous atributs. Per fer-ho, s'han de fer servir dues sentències. En la primera es declara l'atribut indicant-hi el seu nom i el seu tipus de dades. En la segona, s'especifica l'element al qual va associat i el valor que retornarà l'atribut. La sintaxi VHDL per declarar un atribut definit per l'usuari és:

```
attribute nomAtribut : tipusDades;
attribute nomAtribut of nomElement: tipusElement is valorAtribut;
```

Vegem-ne un exemple. Suposem que un usuari desitja crear un atribut anomenat *keep* de tipus booleà associat a un senyal anomenat S i amb valor *true*. La declaració es realitzaria de la manera següent:

```
attribute keep: boolean;
attribute keep of S: signal is true;
```

Quan l'usuari invoqui l'expressió *S'keep*, aquesta retornarà el valor *true*.

6.3. Circuits combinacionals

6.3.1. Declaració d'un sumador complet d'un bit

En aquest apartat es fa una descripció (de baix nivell d'abstracció) d'un disseny combinacional i es mostra que amb un llenguatge descriptiu de maquinari com VDHL es pot fer una descripció equivalent a l'entrada d'esquemàtics clàssica.

Exemple d'un *full adder*
adder.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity adder is
-- i0, i1 i el carry-in ci són entrades de l'adder (sumador).
-- s és la sortida de la suma, co és el carry-out de sortida.
  port (i0, i1 : in bit;
        ci : in bit;
        s : out bit;
        co : out bit);
end adder;

architecture rtl of adder is
begin
-- Aquesta arquitectura de full-adder conte dues assignacions concurrents.
-- Computa la suma.
  s <= i0 xor i1 xor ci;
-- Computa el carry.
  co <= (i0 and i1) or (i0 and ci) or (i1 and ci);
end rtl;
```

Aquesta descripció textual d'un *adder* és equivalent a l'entrada d'esquemes següent.

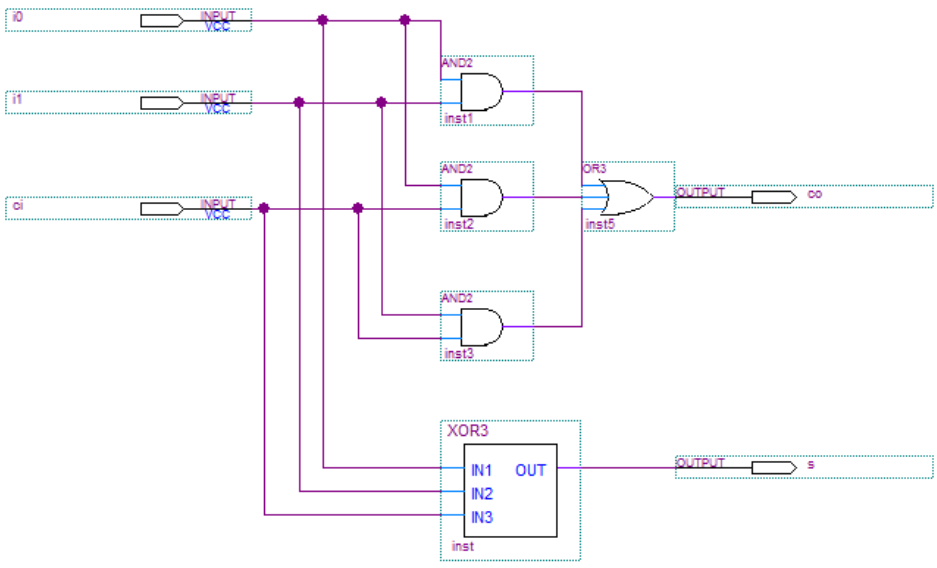


Fig. 6.6 adder.bdf

On la xor3 es pot implementar en Quartus II amb 2 *xor2* de dues entrades:

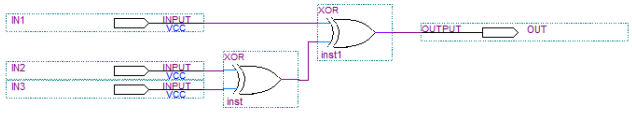


Fig. 6.7 Xor3 a partir de 2 xor2

6.3.2. Declaració d'un sumador de 8 bits a nivell comportamental

A continuació teniu la declaració d'un sumador de 8 bits amb una sortida de 9 bits. Aquesta descripció és molt més potent que l'anterior, ja que la fem a un nivell de descripció més alt i, per tant, més senzill. Si us hi fixeu, només fem C = A + B i ens despreocupem de com l'eina de síntesi ho genera. És a dir, NO hem de fer les operacions *xor* ni *and*/or a nivell de bit.

NOTA IMPORTANT: És molt important que us fixeu en com està implementada l'operació suma. L'operació (*C <= ('0'& A) + ('0' & B)*) té com a entrades dos busos de 8 bits d'entrada que després de l'operació pot donar un resultat de fins a 9 bits, per tant es necessita un bus de 9 bits. Si feu la *connexió* d'un bus de 8 bits amb un de 9 bits l'eina de síntesis (QuartusII) dóna un error. Per solucionar-ho, s'ha de fer el que s'anomena concatenació de bits que consisteix en afegir un bit al bus de 8 bits i això es fa escrivint ('0' & A). Aquesta expressió indica que estem afegint un bit al bus a la part més significativa (MSB, Most Significant Bit) d'A i aquest bit és un 0. Per tant, així obtenim un bus de 9 bits i no dóna l'error.

Adder_beh.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
--------------------------------------------------
entity ADDER_BEH is
  port(
    A : in std_logic_vector(7 downto 0);
    B : in std_logic_vector(7 downto 0);
    C: out std_logic_vector(8 downto 0));
end ADDER_BEH;
--------------------------------------------------
architecture behv of ADDER_BEH is
attribute keep: boolean;
attribute keep of C: signal is true;
begin
  process(A,B)
  begin
    C <= ( '0' & A ) + ('0' & B);
  end process;
end behv;
```

En el codi del sumador, dins l'arquitectura, es defineix *keep* com un atribut de tipus booleà, se li assigna el valor *true* i s'associa al senyal *C*. S'ha de remarcar que la definició de l'atribut *keep* no és necessària per a la descripció VHDL del sumador. S'ha inclòs a tall d'exemple. La seva comprensió us serà útil quan desenvolupeu el processador més endavant.

L'ús de l'atribut *keep* està relacionat amb la síntesi i la simulació. Durant el procés de síntesi d'un circuit es du a terme una etapa d'optimització que pot provocar la reorganització dels senyals interns del circuit. Conseqüentment, un senyal intern que forma part de la descripció del circuit pot desaparèixer. Això té com a conseqüència que el senyal desaparegut no serà observable en els resultats de la simulació del circuit.

L'atribut *keep* és un atribut que té un significat predefinit per a l'eina de síntesi de Quartus. Quan un senyal combinacional té associat l'atribut *keep* el senyal no es veu afectat per l'optimització que fa l'eina. D'aquesta manera, l'activitat del senyal queda registrada durant la simulació del circuit i pot ser examinada posteriorment.

En incloure la definició de l'atribut *keep* vinculat al senyal *C*, s'està assegurant que, si se sintetitza amb Quartus un circuit que contingui aquest sumador, quan se simuli el circuit es podrà veure el comportament de la sortida *C* del sumador, ja que aquest senyal no haurà estat alterat durant el procés de síntesi.

6.3.3. Multiplexor de 2 entrades: utilització de la sentència if

L'exemple que es mostra a continuació és un multiplexor. Aquest té dues entrades, *a, b* i el selector *sel,* i una sortida *f*. A sota es mostra el codi. Fixeu-vos que a la llista de sensibilitat (*process*) hi ha totes les entrades: *a, b, sel*.

MUX.vhd

```
library IEEE;
entity MUX is
  port (a, b, sel: in bit;
        f : out bit);
end MUX;

architecture rtl of MUX is
begin
  process (sel, a, b) is
  begin
    if (sel = '1') then
      f <= a;
    else
      f <= b;
    end if;
  end process;
end rtl;
```

6.3.4. Taules de conversió: Utilització de la sentència case

El case és una manera senzilla d'evitar llargues i confoses imbricacions (*nesting*) dels ELSE-IFs .

```
[ label: ] case <expressió> is
  when <elecció1> =>
    <sentències seqüencials a realitzar>
  when <elecció2> =>      -- opcional
    <sentències seqüencials a realitzar>
  ...
  when others =>  -- obligatori si totes les altres opcions no estan cobertes
    <sentències seqüencials a realitzar>
end case [ label ];
```

Per exemple:

```
case my_val is
  when 1 =>
    a <= b;
  when 3 =>
    c <= d;
  when others =>
    null;
end case;
```

L'exemple següent mostra la descripció de maquinari a través d'una taula de conversió. Per exemple, un conversor bcd – 7 segments en VHDL. El circuit té els valors en *bcd* (*binary-coded decimal*) com a entrades i aquest dóna els valors necessaris per il·luminar el càtode del *display* de 7 segments. Per a valors no vàlids a l'entrada *bcd*, només s'il·luminarà el segment "g". La localització dels diferents segments es mostra a la Fig. 6.8 de més avall. S'hi pot observar que la descripció del circuit s'aconsegueix bàsicament amb una sentència *case*.

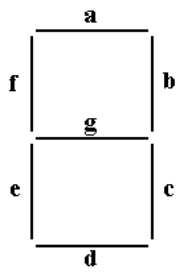


Fig. 6.8 Disposició dels diferents leds en un 7 segments

bcd27seg.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity bcd27seg is
  port (
    bcd: in std_logic_vector(3 downto 0);
    segment: out std_logic_vector(6 downto 0));
end bcd27seg;

architecture design of bcd27seg is
begin
  process (bcd)
  begin
    case bcd is       -- ordre dels segments del display: a b c d e f g
      when "0000" => segment <= "1111110";
      when "0001" => segment <= "0110000";
      when "0010" => segment <= "1101101";
      when "0011" => segment <= "1111001";
      when "0100" => segment <= "0110011";
      when "0101" => segment <= "1011011";
      when "0110" => segment <= "1011111";
      when "0111" => segment <= "1110000";
      when "1000" => segment <= "1111111";
      when "1001" => segment <= "1110011";
      when others => segment <= "0000001";
    end case;
  end process;
end design;
```

A sota es mostra la simulació en Quartus II del 7 segments. Només cal comentar que per valors no vàlids del 7 segments s'il·lumina exclusivament el segment "g".

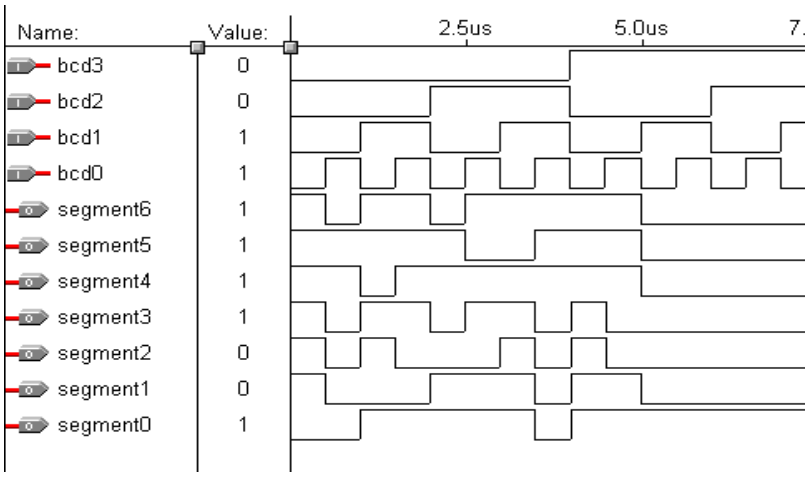


Fig. 6.9 Formes d'ona d'un conversor bcd -7 segments

6.4. Mòduls d'emmagatzemament d'informació

La unitat bàsica d'emmagatzemament d'un registre és una balda (*latch*) o bé un *flip-flop*. La diferència entre l'un i l'altre és que el *latch* actua per nivell i el *flip-flop* per flanc. Ambdós és poden dir biestables, ja que tenen 2 estats: el '0' lògic i l''1' lògic.

6.4.1. Descripció d'un latch

A l'exemple següent es mostra la descripció d'un D-l*atch*. És interessant veure-hi l'ús de la sentència d'assignació concurrent condicional (*when ..else*).

DLatch.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity dlatch is
  port (D, ENA: in std_logic;        -- D dada, ENA habilitació
        Q, QN: out std_logic);       -- Sortida Q i sortida negada QN
end dlatch;

architecture behave of dlatch is
signal state : std_logic;            -- Aquest senyal guarda la dada
begin
  state <= D when ENA = '1' else     -- l'estat canvia només si ENA=1
           state;
  Q <= state;                        -- Q és state
  QN <= not state;                   -- Q és state negat
end behave;
```

6.4.2. Descripció d'un flip-flop

A l'exemple següent es mostra la descripció d'un *flip-flop* E. Un *flip-flop E* és molt similar a un *flip-flop* D però el *flip-flop E* té, a més, una entrada d'*enable* que habilita quan l'entrada D proporciona la informació al *flip-flop* i quan no. Aquest *flip-flop* té una entrada d'habilitació (*ENABLE*) síncrona i un *RESET* asíncron. Fixeu-vos que a la descripció del *flip-flop* l'efecte de *RESET* està abans de l'actuació del rellotge i, per tant, no depèn d'aquest. En canvi, el senyal *ENABLE* sí que està regit pel comportament del rellotge.

EFlipFlop.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Eflipflop is       -- El flip-flop és sensible al flanc del rellotge
  port (D : in std_logic;          -- Entrada D del flipflop
        ENABLE : in std_logic;     -- Habilitació de l'entrada
        CLK :in std_logic;         -- rellotge del sistema
        RESET: in std_logic ;      -- RESET del sistema
        Q, QN: out std_logic );    -- Sortida Q i QN sortida negada
end Eflipflop;

architecture behave of Eflipflop is
begin
  process(CLK,RESET) is            -- començament del process
  -- codi sequencial
  -- reactiu a canvis en CLK, RESET, ENABLE i en D
  variable state : std_logic;      -- variable on s'emmagatzema l'estat
  begin
    if(RESET = '1') then           -- RESET asíncron
      state := '0';
    elsif(CLK'EVENT AND CLK='1') then
      if(ENABLE = '1') then        -- ENABLE síncron
        state := D;
      end if;
    end if ;
    Q <= state;
    QN <= not state;
  end process;
end behave;
```

6.5. Sistemes seqüencials

6.5.1. Comptador

El llenguatge VHDL mostra la seva potència a l'hora de descriure comptadors i altres sistemes seqüencials com es mostra una mica més endavant. Considereu el comptador mòdul 200 que es mostra a continuació:

Modulo200.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;

entity modulo200 is
  port (CLRN, CLK : in std_logic ;
        Q: out std_logic_vector (7 downto 0));
end modulo200;

architecture rtl of modulo200 is
begin
  -- comptador incrementador (up counter)
  process(CLK, CLRN)
  variable cnt : integer range 0 to 255;
  constant modulus : integer range 0 to 255 := 200;
  begin
    if(CLRN = '0') then
      cnt := 0;
    elsif (CLK'EVENT AND CLK = '1') then
      if cnt = modulus - 1 then
        cnt := 0;
      else
        cnt := cnt + 1;
      end if;
    end if;
    Q <= conv_std_logic_vector(cnt,8);
  end process;
end rtl;
```

Podeu veure que, a partir de la descripció en un codi d'unes 25 línies, tenim un comptador mòdul 200. Aquest pot ser modificat per obtenir altres tipus de comptadors canviant rang, mòdul, longitud del vector, sentit, valor inicial, etc.

6.5.2. Màquines d'estats finits en VHDL

En aquest apartat s'expliquen diferents tipus de màquines d'estats, la seva representació gràfica i la manera de descriure-les en VHDL. Ens centrem només en autòmats síncrons. Normalment cada màquina d'estats pot ser descrita com un o dos processos separats. Més endavant s'expliquen breument els avantatges i els inconvenients de la descripció d'un autòmat amb un o dos processos. En una màquina d'estats cada estat té un nom, el qual hauria d'indicar allò que caracteritza aquest estat. Això s'aconsegueix en VHDL utilitzant un tipus enumerat, els elements del qual són els noms dels estats. Després, en el procés de síntesi, aquests estats s'implementen mitjançant una representació binària. Aquest procés s'anomena codificació dels estats.

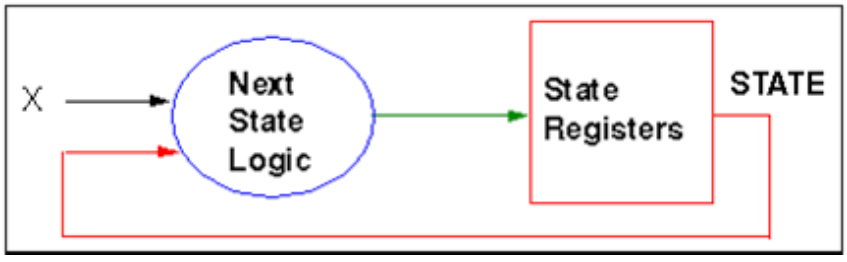
Hi ha diferents tipus de màquines d'estats finits. Els tipus més coneguts són màquines de Moore i màquines de Mealy, encara que n'hi ha d'altres.

6.5.2.1. Descripció d'una màquina d'estats amb 1 procés (ús exclusiu de l'estat actual)

A la figura 6.10 es mostren tres maneres de representar una màquina d'estats finits. La gràfica de més amunt és un diagrama que conté només els blocs més rellevants i els senyals d'interès (les sortides de la màquina s'han omès). El primer bloc (oval) representa la lògica combinacional de l'autòmat, i el segon (rectangle), els elements d'emmagatzemament.

El gràfic de sota a la dreta descriu l'autòmat com un diagrama d'estats. Si la condició que connecta cadascuna de les transicions (fletxes) avalua a veritat en el moment que el rellotge està actiu, llavors l'autòmat canvia el seu estat. Això passa d'una manera síncrona. Aquí el *reset* asíncron és l'única excepció a aquest comportament regit pel rellotge. En el moment que el senyal de *reset* està actiu, l'autòmat canvia a l'estat *START* de manera immediata.

A la part inferior esquerra de la figura, hi ha la corresponent descripció en llenguatge VHDL. L'autòmat està descrit per un procés que depèn del rellotge i del *reset*. La primera branca de l'IF conté la descripció de l'efecte del *reset*. A la branca següent, la branca *ELSIF*, es descriu la resta de l'autòmat. A la declaració del *CASE*, el qual modela la transició entre estats, l'estat actual de l'autòmat és detectat i s'examina si els valors d'entrada són presents per poder fer un canvi d'estat.




```
FSM_FF: process(CLK, RESET)
begin
  if RESET = '1' then
    STATE <= START;
  elsif CLK'event and CLK = '1' then
    case STATE is
      when START  => if X = GO_MID then
                       STATE <= MIDDLE;
                     end if;
      when MIDDLE => if X = GO_STOP then
                       STATE <= STOP;
                     end if;
      when STOP   => if X = GO_START then
                       STATE <= START;
                     end if;
      when others => STATE <= START;
    end case;
  end if;
end process FSM_FF;
```

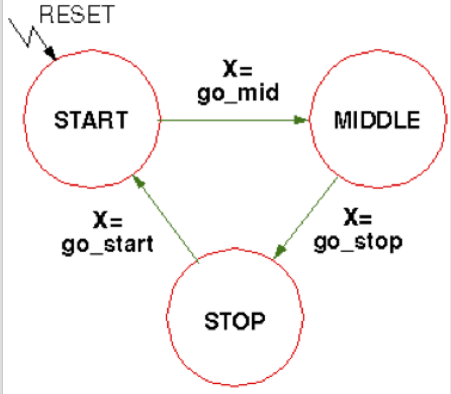
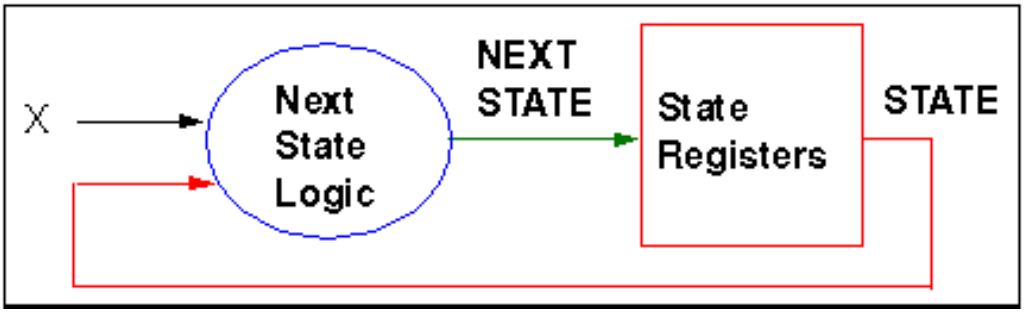


Fig. 6.10 Diferents representacions d'una màquina d'estats. La descripció en VHDL es fa amb un sol procés

6.5.2.2. Descripció d'una màquina d'estats amb dos processos (ús d'estat actual i estat següent)

El mateix autòmat s'utilitza per mostrar la implementació basada en dos processos VHDL (Fig. 6.11). El senyal *NEXT_STATE* s'examina aquest cop explícitament. S'insereix en el diagrama de blocs entre la lògica combinacional i els elements d'emmagatzemament. En el diagrama d'estats no hem hagut de fer cap canvi, el comportament és el mateix.

El codi VHDL font conté dos processos. Hi ha un procés comandat pel rellotge que descriu els elements de memòria i un altre procés que descriu la lògica combinacional. En la declaració del *CASE*, un altre cop, l'estat actual es comprova i els valors d'entrada s'examinen. Si l'estat ha de canviar, llavors *NEXT_STATE* i *STATE* difereixen. El proper cop que hi ha un flanc actiu de rellotge, aquest nou estat es pren com a estat actual.




```
FSM_FF: process (CLK, RESET)
begin
  if RESET = '1' then
    STATE <= START;
  elsif CLK'event and CLK='1' then
    STATE <= NEXT_STATE;
  end if;
end process FSM_FF;

FSM_LOGIC: process (STATE, X)
begin
  NEXT_STATE <= STATE;
  case STATE is
    when START  => if X = GO_MID then
                     NEXT_STATE <= MIDDLE;
                   end if;
    when MIDDLE => if X = GO_STOP then
                     NEXT_STATE <= STOP;
                   end if;
    when STOP   => if X = GO_START then
                     NEXT_STATE <= START;
                   end if;
    when others => NEXT_STATE <= START;
  end case;
end process FSM_LOGIC ;
```

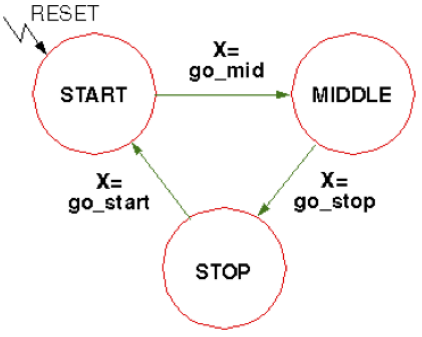


Fig. 6.11 Diferents representacions d'una màquina d'estats. La descripció VHDL es fa amb dos processos.

Estructura i llegibilitat

El model en VHDL ha de representar el maquinari, el qual ha de ser creat a partir de la descripció del codi font del VHDL. Per tant, l'estructura ha de ser vista, d'una banda, com a lògica purament combinacional i, de l'altra, com una estructura que desa l'estat. Això són dues estructures diferents.

Simulació

És més fàcil de detectar possibles errors del model VHDL en la forma d'ones si es té accés als senyals intermedis *NEXT_STATE*. Per tant, el temps i la localització on es cometen els errors per primer cop poden ser determinats més acuradament. La versió amb dos processos és millor en aquest sentit.

Síntesi

Els algoritmes de síntesi estan basats en heurística. Per tant, és possible donar una declaració universal vàlida. Però, diverses eines de síntesi tendeixen a produir diferents resultats (si no tenim en compte sofisticats *scripts*). La descripció de l'autòmat en dos processos tendeix a produir un millor resultat de síntesi, ja que la seva descripció està més propera a l'estructura que té el *hardware*.

6.5.2.3. Màquines d'estats finits de Moore

Les màquines de Moore són un tipus de màquines d'estats en les quals l'estat present depèn només de les entrades anteriors i els estats anteriors, i la sortida actual depèn només de l'estat actual.

A la Fig. 6.12 es mostra un exemple de màquina de Moore, on el valor dels vectors de sortida és funció de l'estat actual. Aquesta és la raó per la qual el segon bloc lògic en el diagrama de blocs està situat després dels elements de memòria. Aquest bloc té el *hardware* necessari per calcular els valors de sortida a partir de l'estat actual (*STATE*) de l'autòmat.

En el codi font de VHDL, aquesta lògica s'implementa amb un procés específic. Com que els valors de sortida depenen dels valors de l'estat actual, no es necessiten més senyals que els que apareixen a la llista de sensibilitat del procés. A la Fig. 6.13 queda esquematitzat com es faria la descripció d'una màquina de Moore amb dos i tres processos, respectivament. Per la seva banda a la Fig. 6.14 es pot observar un exemple de màquina de Moore descrita mitjançant tres processos. De fet, és interessant observar que en aquest exemple, en lloc del tercer procés, que implementaria la lògica combinacional que genera les sortides, s'utilitzen dues sentències d'assignació concurrent condicional.

Màquina de Moore

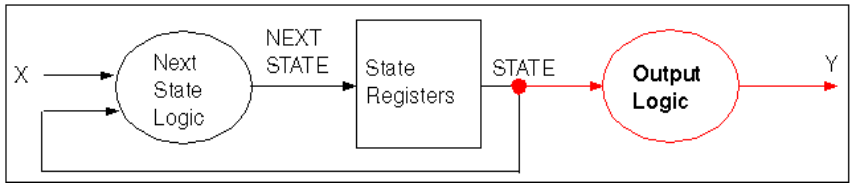


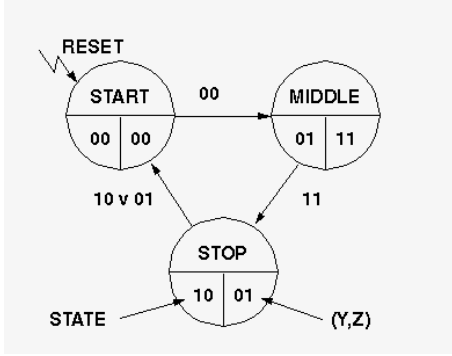
Fig. 6.12 El vector de sortida és funció de l'estat de la màquina: Y = f(STATE)

Tres Processos	Dos Processos

`architecture RTL of MOORE is`
`...`
`begin`
`  REG: -- Clocked Process`
 	
`  CMB: -- Combinational Process`
 	
`  OUTPUT: process (STATE)`
`  begin`
`  -- Output Logic`
`  end process OUTPUT ;`
`end RTL ;`
	`architecture RTL of MOORE is`
	`...`
	`begin`
	`  REG: process (CLK, RESET)`
	`  begin`
	`  -- State Registers Inference`
	`  -- with Next State Logic`
	`  end process REG;`
	
	`  OUTPUT: process (STATE)`
	`  begin`
	`  -- Output Logic`
	`  end process OUTPUT ;`
	`end RTL;`

Fig. 6.13 Màquina de Moore amb 2 i 3 processos.

Codificació en VHDL
Moore.vhd




```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity MOORE is
  port (RESET,CLK:in std_logic;
        A,B : in std_logic;
        Y,Z: out std_logic);
end MOORE;

architecture RTL of MOORE is
type STATE_TYPE IS (STA, MIDDLE, STP);
signal STATE,NEXTSTATE : STATE_TYPE;
begin
  REG: process (CLK, RESET)
  begin
    if RESET='1' then
      STATE <= STA;
    elsif CLK'event and CLK='1' then
      STATE <= NEXTSTATE;
    end if;
  end process REG;

  CMB: process (A, B, STATE)
  begin
    NEXTSTATE <= STATE;
    case STATE is
      when STA    => if (A or B)='0' then
                       NEXTSTATE <= MIDDLE ;
                     end if;
      when MIDDLE => if (A and B)='1' then
                       NEXTSTATE <= STP ;
                     end if;
      when STP    => if (A xor B)='1' then
                       NEXTSTATE <= STA ;
                     end if;
      when others => NEXTSTATE <= STA ;
    end case;
  end process CMB ;

-- aquestes dues sentències d'assignació concurrent condicional
-- equivalen al tercer procés
  Y <= '1' when STATE=MIDDLE else
       '0';
  Z <= '1' when STATE=MIDDLE or STATE=STP else
       '0';
end RTL ;
```

Fig. 6.14 Exemple de màquina de Moore (diagrama d'estats i codi VHDL)

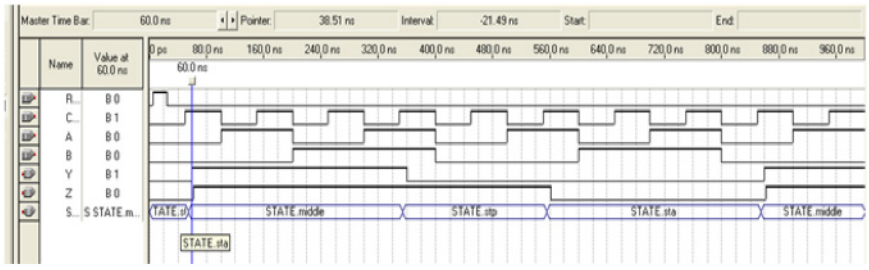
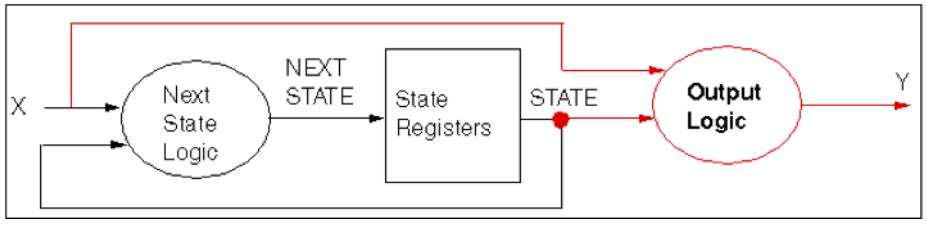


Fig. 6.15 Formes d'ona de la màquina de Moore

6.5.2.4. Màquines d'estats finits de Mealy

Les màquines de Mealy són un tipus de màquines d'estats en què les sortides depenen de l'estat actual i de les entrades.

A la Fig. 6.16 es mostra un autòmat de Mealy. El valor de les sortides és una funció del valor de l'estat actual i dels valors de les entrades: Y = f(X, STATE). Per això hi ha dibuixada una línia al diagrama de blocs des de l'entrada fins al bloc que calcula els valors de sortida. A la part inferior de la figura queda esquematitzat com es faria la descripció d'una màquina de Mealy amb dos i tres processos, respectivament. Recordeu que en el codi font de VHDL, els senyals d'entrada han d'estar a la llista de sensibilitat del procés que li correspon. A la Fig. 6.17 hi ha un exemple de màquina de Mealy descrita mitjançant tres processos, amb la particularitat que hi ha dues sentències d'assignació concurrent condicional en lloc del tercer procés.



<u>Tres Processos</u>

```
architecture RTL of MEALY is
...
begin
  REG: -- Clocked Process

  CMB: -- Combinational Process

  OUTPUT: process (STATE, X)
  begin
  -- Output Logic
  end process OUTPUT ;
end RTL ;
```

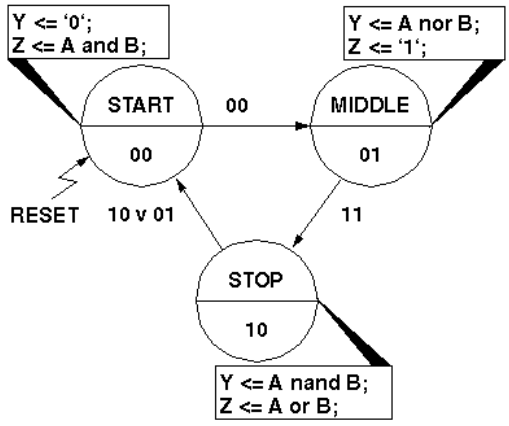
<u>Dos Processos</u>

```
architecture RTL of MEALY is
...
begin
  MED: process (CLK, RESET)
  begin
  -- State Registers Inference with
  -- Next State Logic
  end process MED;

  OUTPUT: process (STATE, X)
  begin
  -- Output Logic
  end process OUTPUT;
end RTL ;
```

Fig. 6.16 Màquina de Mealy amb 2 i 3 processos.

Codi VHDL ***Mealy.vhd***




```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity MEALY is
  port (RESET,CLK:in std_logic;
        A,B : in std_logic;
        Y,Z: out std_logic);
end MEALY;

architecture RTL of MEALY is
type STATE_TYPE IS (STA, MIDDLE, STP);
signal STATE,NEXTSTATE : STATE_TYPE;
begin
  REG: process (CLK, RESET)
  begin
    if RESET = '1' then
      STATE <= STA ;
    elsif CLK'event and CLK='1' then
      STATE <= NEXTSTATE ;
    end if ;
  end process REG;

  CMB: process (A,B,STATE)
  begin
    -- Like MOORE Examples
    NEXTSTATE <= STATE ;
    case STATE is
      when STA    => if (A or B) = '0' then
                       NEXTSTATE <= MIDDLE ;
                     end if ;
      when MIDDLE => if (A and B) = '1' then
                       NEXTSTATE <= STP ;
                     end if ;
      when STP    => if (A xor B) = '1' then
                       NEXTSTATE <= STA ;
                     end if;
      when others => NEXTSTATE <= STA ;
    end case ;
  end process CMB ;

-- concurrent signal assignments for output
  Y <= '1' when (STATE=MIDDLE and (A or B)='0') or
                (STATE=STP and (A and B)='0')      else
       '0';
  Z <= '1' when (STATE=STA and (A and B)='1') or
                (STATE=MIDDLE) or (STATE=STP and (A or B)='1') else
       '0';
end RTL ;
```

Fig. 6.17 Exemple de màquina de Mealy (diagrama d'estats i codi VHDL)

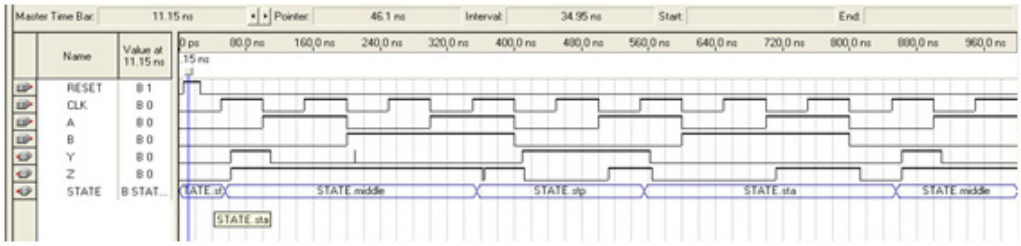


Fig. 6.18 Formes d'ona de la màquina de Mealy

6.6. Blocs elementals de LittleProc en VHDL

A continuació expliquem els mòduls que s'han de fer en VHDL que són parts integrants del microprocessador que farem.

En capítols posteriors incorporareu aquests mòduls al processador LittleProc. Aleshores, quan simuleu el processador us interessarà examinar el comportament de certs senyals combinacionals del mòduls (per exemple, la sortida *C* de l'ALU o les sortides dels mòduls combinacionals). Durant l'etapa de síntesi del processador, l'optimització del circuit que es du a terme pot provocar la desaparició dels senyals combinacionals abans esmentats, i fer imposible la seva observació durant la simulació del processador. Per evitar aquest fet, recordeu que en el codi VHDL, dins l'apartat declaratiu de la sentència *architecture*, heu d'associar l'atribut *keep* als senyals combinacionals que voleu observar. Vegem-ne un exemple:

```
attribute keep: boolean;
attribute keep of nomSenyal: signal is true;
```

6.6.1. Realització d'un incrementador

S'ha de fer un mòdul incrementador (inc). Penseu que aquest mòdul és una versió simplificada d'un sumador, que només fa l'operació amb un operand A produint el resultat A+1. Aquest mòdul ha de tenir 8 bits d'entrada i 8 bits de sortida (Fig. 6.19).

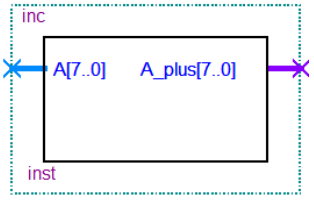


Fig. 6.19 Símbol del mòdul inc

6.6.2. Realització d'un registre

Aquest registre és diu *reg* i és actiu per flanc, NO per nivell.

Entrades:

asynResetn: Té un bit per fer un *reset* asíncron, actiu a baixa.
ck: Un senyal del rellotge *ck*, actiu per flanc de pujada.
load: Un bit de càrrega de la dada síncrona; *load* actiu a alta.
datain[11..0]: El registre té un bus d'entrada de la dada *(*datain) de 12 bits.

Sortides:

dataout[11..0]: El mòdul té un bus de sortida (dataout) també de 12 bits.

A continuació es mostra el símbol del bloc *reg*.

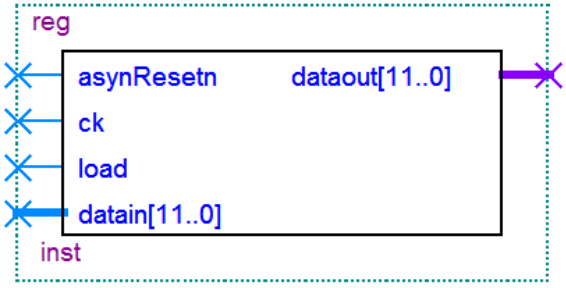


Fig. 6.20 Símbol del mòdul reg

6.6.3. Realització de l'ALU (Unitat Aritmeticològica)

Funcionalitat i Terminals

S'ha de realitzar una unitat aritmeticològica que faci les operacions següents: pas transparent (C<=B), resta (C<=A-B), suma (C<=A+B), màscara dels quatre primers bits (C<=B and 0FF), incrementació d'una unitat (C<=B+1), rotació a la dreta i desplaçament a la dreta, ambdues amb inclusió del *carry*. Per tal de fer això, aquest bloc té les entrades i sortides següents:

Entrades:

A[11..0] i B[11..0]: 2 busos d'entrada de dades de 12 bits (A, B).
sel[2..0]: Un bus de selecció d'operació de 3 bits (sel).
ck: Una entrada de rellotge per sincronitzar l'actualització dels senyals de condició carry, zero i negative.

Sortides:

C[11..0]: Bus de sortida de 12 bits (C).
carry, zero i negative: Senyals de condició, indicadors de certes propietats dels resultats de les operacions.

Un diagrama d'entrada sortida de la unitat aritmeticològica es mostra a la Fig. 6.21.

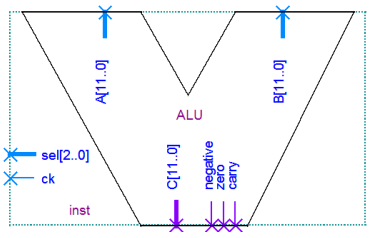


Fig. 6.21 Símbol de la unitat aritmeticològica

Processos

La unitat aritmeticològica té dos processos diferenciats. A la Fig. 6.22 es mostra un diagrama de com podria ser aquesta unitat. Penseu que hi ha diverses maneres de fer aquesta implementació i que aquí només se n'explica una manera.

Procés 1

Aquest procés contindrà una sentència CASE on es descriurà cadascuna de les operacions a realitzar segons el valor del senyal *sel*.

Cas "000" realitza el pas transparent de les dades del bus B. Penseu que probablement necessiteu fer una concatenació de les dades.

Cas "001" realitza la resta C = A – B; per fer aquesta operació s'ha de fer el complement a 2 de la dada B. És a dir, fer una operació tal que al final C adopti el valor [(A) + ((not (B)) + 1];

Cas "010" realitza la suma C = A + B;

Cas "011" realitza la màscara dels primers 4 bits de l'entrada B; és a dir, C al final ha d'adoptar el valor B and 0FF;

Cas "100" incrementa en 1 el valor que entra pel bus B; per tant C rebrà el valor B + 1;

Nota important: penseu que RESULTAT és recomanable que sigui de 13 bits; per tant, heu de fer concatenacions (&) en els busos A i B per obtenir 13 bits.

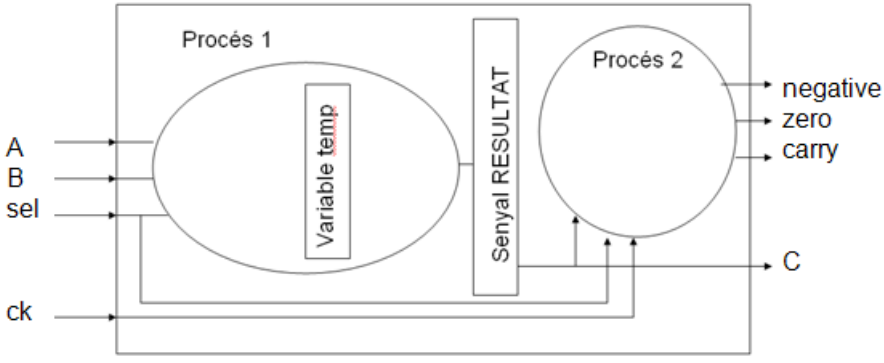


Fig. 6.22 Possible diagrama de la unitat aritmeticològica.

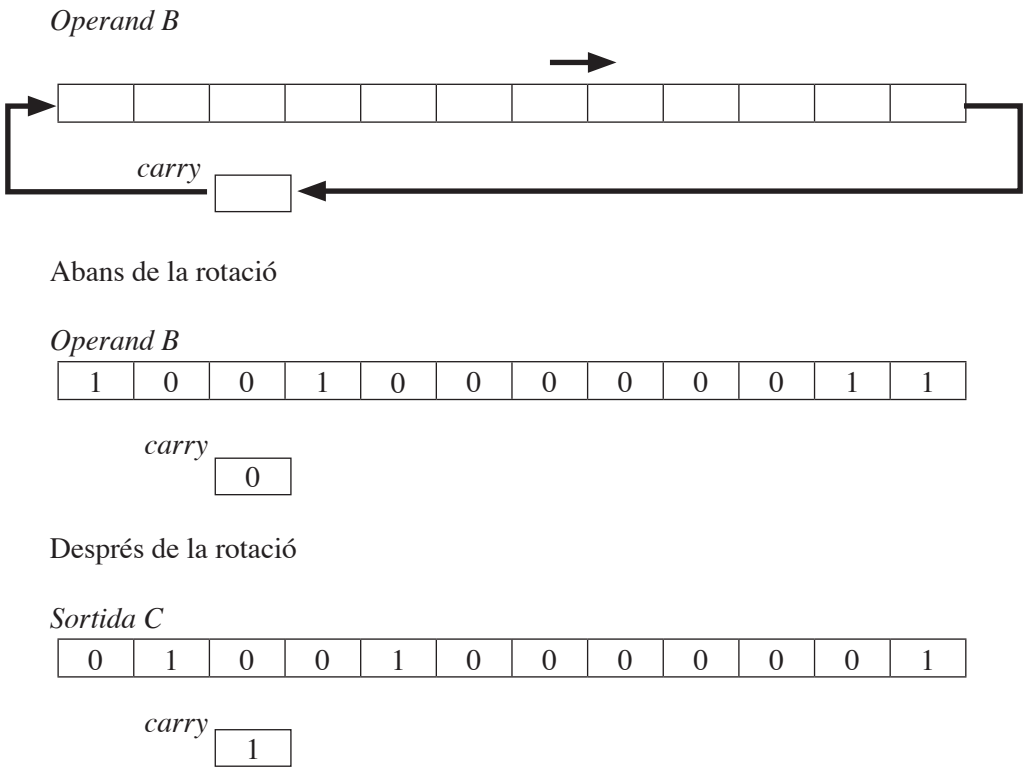


Fig. 6.23 Rotació a la dreta incloent-hi el *carry*.

Cas "101" fa la rotació a la dreta incloent-hi el carry (vegeu la Fig. 6.23). El seu codi es dóna com a exemple més avall. La funció *to_bitvector* fa que un valor *std_logic signed* o *unsigned* com per exemple B pugui ser transformat a un *bit_vector* utilitzant aquesta conversió; això es fa perquè *srl* només es pot operar amb *bit_vectors*. Un cop feta l'operació es fa un altre cop la conversió cap a *std_logic_vector* amb la funció *to_stdlogicvector.*

```
RESULTAT <=('0' & (to_stdlogicvector(to_bitvector(B) srl 1 ))) ;
RESULTAT(12) <= B(0);
RESULTAT(11) <= carryV;
```

Cas "110": tenint l'exemple del *cas "101"* feu el desplaçament a la dreta incloent-hi el carry (vegeu la Fig. 6.24).

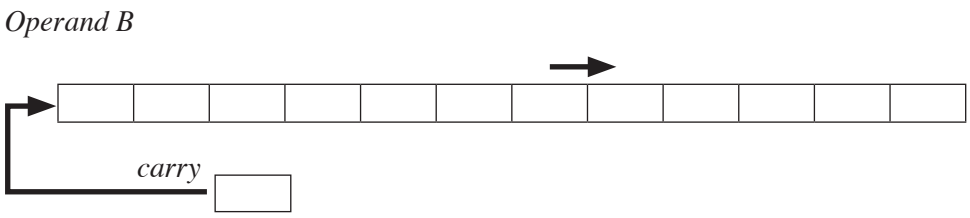
Procés 2

L'altre procés registra els bits de sortida *zero, carry* i *negative*. Com que aquests han de ser *flip-flops*, han d'estar controlats pel rellotge *ck* (flanc actiu de pujada).

Els senyals *zero*, *carry* i *negative* es veuen afectats per totes les operacions a excepció del pas transparent i de l'emmascarament. A més, el senyal zero tampoc no queda afectat per l'operació d'increment.

És important tenir en compte que el senyal *zero* és actiu quan el senyal intern RESULTAT és 0 en els seus 12 últims bits.

Els senyals de sortida *negative* i *carry* no deixen de ser només uns bits del bus *RESULTAT*.



Abans del desplaçament

Operand B

1	0	0	1	0	0	0	0	0	0	1	1

carry

0

Després del desplaçament

Sortida C

0	1	0	0	1	0	0	0	0	0	0	1

carry

0

En el desplaçament el *carry* no varia.

Fig. 6.24 Desplaçament a la dreta incloent-hi el *carry*

Un possible procés de registració dels *flags* seria (nota: el senyal *carryV* és un senyal auxiliar que sempre té el mateix valor que *carry)*:

```
process (<???????>)
begin
  if(ck'event and ck='<?>') then
    if(sel="<???>" ... ) then
      if(sel /= "<???>" ) then           -- cas especial del senyal zero
        if(RESULTAT(<??> downto <?>) = "<???????????>") then
          zero <= '<?>';
        else
          zero <= '<?>';
        end if;
      end if;
      negative <= '<?>';
      carry <= '<?>';
      carryV <= '<?>';
    end if;
  end if;
end process;
```

NOTA : <????> són parts a omplir.

Per tal de facilitar la implementació de les operacions, a més de les biblioteques habituals hi heu d'afegir aquesta:

```
use ieee.numeric_bit.all;
```

6.6.4. Implementació del bloc IRdecoder

S'ha de fer un conversor que tingui la taula següent:

Bus d'entrada: opcode
Bus de sortida: initAddress

Segons els valors d'entrada, a la sortida s'han de veure els valors mostrats a la taula de la Fig. 6.25. En aquesta figura podeu observar també el símbol de l'IRdecoder. Quins valors anirien en lloc dels interrogants que hi apareixen?

opcode	initAddress
0000	00001000
0001	00001101
0010	00010010
0011	00010101
0100	00011001
0101	00011100
0110	00011111
0111	00100001
1000	00101000
1001	00000111
1010	00100100
Altres casos	XXXXXXXX

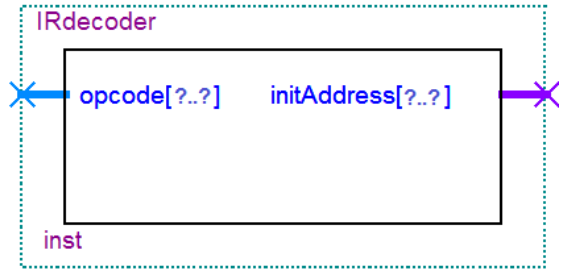


Fig. 6.25 Taula de conversió i símbol del bloc IRdecoder

7. Arquitectura interna del processador

7.1. Introducció

Tal com s'ha anat observant als darrers apartats, les propostes de disseny realitzades anaven destinades a dissenyar certes parts d'un petit processador mitjançant el disseny per captura esquemàtica i el disseny amb VHDL.

Així doncs, el resultat de la integració de totes les petites parts ja realitzades, així com les parts que presentarem en aquesta secció, juntament amb una certa arquitectura, donen lloc al disseny d'un processador elemental adaptable a les nostres necessitats específiques.

7.2. Característiques genèriques del processador

El processador que dissenyarem, anomenat *LittleProc*, té les especificacions tècniques següents:

— Processador de 12 bits.
— Freqüència de rellotge: 10 MHz (limitat a la tecnologia de la FPGA)
— Arquitectura Von Newmann amb unitat de procés de dos busos.
— Memòria de dades/programa: 128 paraules (*Words)* escalable a *4096* paraules
— Unitat de control amb seqüenciador.
— Conjunt d'instruccions (*instruction set*) inicial de 12 instruccions.

7.3. Interfície del processador

El processador que estem dissenyant té la interfície següent amb l'exterior:

Entrades:

start, actiu a alta, que permet posar en funcionament el processador.
asynResetn, actiu a baixa, que permet fer un rèset de tota l'arquitectura.
ck, és el rellotge global de tot el processador i té una freqüència de 10Mhz.

Sortides:

end, actiu a alta. S'activa quan el processador ha acabat l'execució d'un programa.

A la Fig. 7.1 n'observem el símbol.

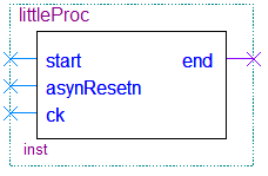


Fig. 7.1 Pinout extern del processador

7.4. Arquitectura interna del processador

De la mateixa manera que la resta de processadors del mercat, l'arquitectura interna de la nostra CPU està formada pels elements següents:

Una memòria RAM de dades/programa, que conté el programa a executar i una zona de memòria reservada per a dades (variables).

Una unitat de control que és l'encarregada de generar la seqüència d'ordres necessàries per executar cada instrucció emmagatzemada a la RAM interna.

La unitat de procés, que és l'encarregada de processar totes les dades, seguint les ordres de la unitat de procés.

Un seguit de *buffers* que permetin la monitorització correcta de les dades per fer la simulació final; aquests no seran necessaris per a la implementació final.

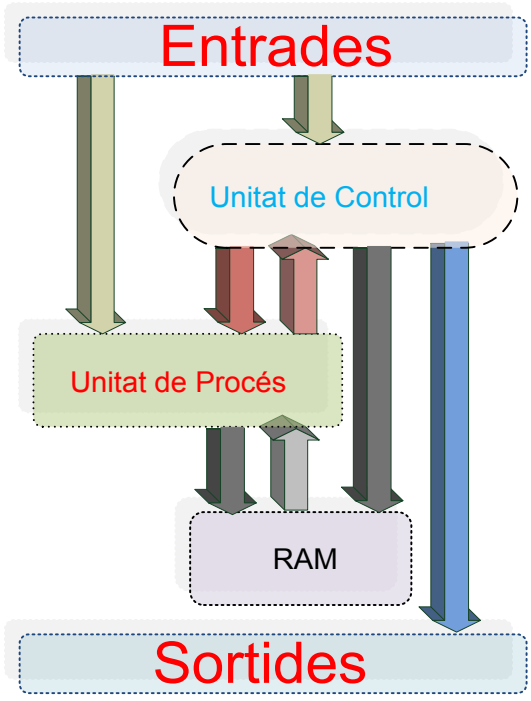


Fig. 7.2 Estructura interna del processador

7.5. Disseny de la unitat de procés de dos busos

Com ja s'ha vist anteriorment, la unitat de procés té la funció d'emmagatzemar el conjunt de registres que s'encarreguen del funcionament correcte de l'execució d'instruccions, així com de realitzar les operacions necessàries.

Està formada per:

— Un conjunt de registres d'emmagatzemament
— Un multiplexor
— 2 busos
— Una unitat aritmeticològica (ALU)

Al diagrama següent en podem observar l'estructura interna.

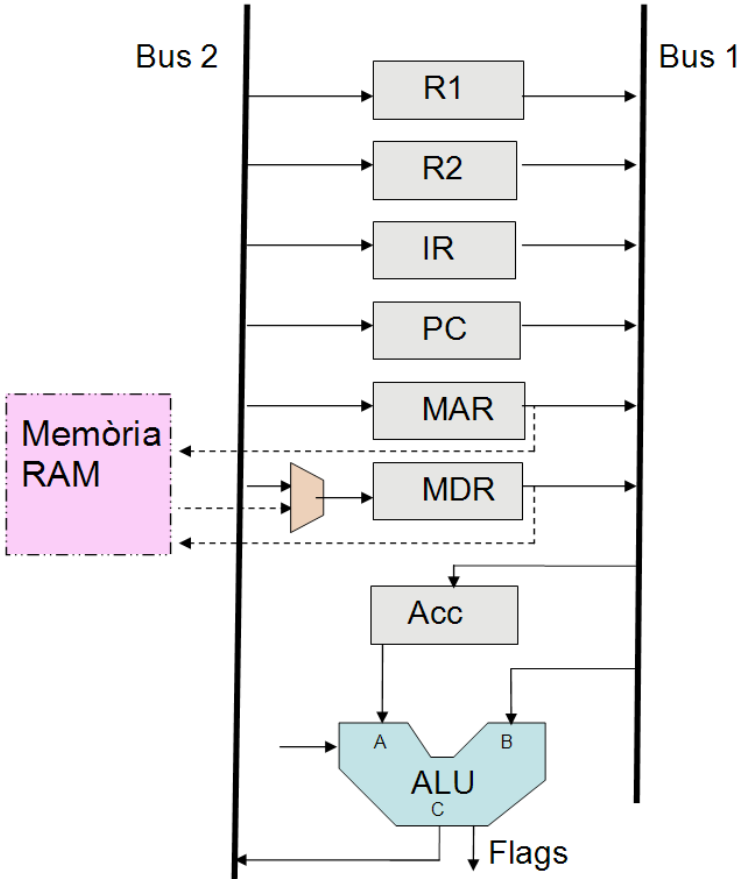


Fig. 7.3 Estructura interna de la unitat de procés

7.5.1. Descripció de la funcionalitat de cada element

Tot seguit fem una descripció funcional de les parts internes de la unitat de procés, tot descrivint:

— Busos interns
— Funcionalitat de cada registre
— Unitat aritmeticològica

7.5.1.1. Bus 1

És el bus de sortida dels registres interns de la unitat de procés i està format per un total de 12 bits. Aquest bus l'anomenem Bus1[11..0].

Té la funció de proporcionar a l'ALU un operand. També proporciona dades a l'acumulador (per tal de proporcionar dos operands a les instruccions que així ho requereixin).

Com que a aquest bus accedeixen alhora totes les sortides dels registres (excepte l'acumulador) és necessari posar un *buffer tristate* múltiple a la sortida de cada registre per garantir que només un d'ells accedeixi al bus, i no faci un curtcircuit.

Aquests *buffers* són controlats (cadascun) per uns senyals d'entrada de la unitat de procés (UP). El component d'Altera s'anomena Tri, i el pin que permet controlar-lo s'ha d'anomenar *NomRegistreOut*. D'aquesta manera, a mode d'exemple, per al registre R2 aquest senyal s'hauria d'anomenar R2out (vegeu Fig. 7.4).

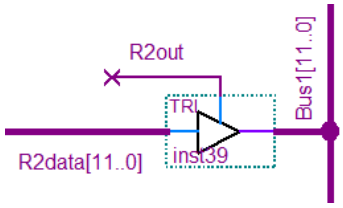


Fig. 7.4 Buffer tri-state per a la sortida del registre R2

7.5.1.2. Bus 2

Aquest bus té l'objectiu principal de subministrar dades (que majoritàriament provenen de l'ALU) als registres interns de la UP.

Està format per un total de 12 bits i l'anomenem Bus2[11..0]. Tots els registres excepte l'acumulador i el MDR reben directament les dades del bus, les quals són carregades mitjançant el corresponent senyal de càrrega o *load* anomenat *NomRegistreload*.

Un cas particular i excepcional és el registre MDR, que pot rebre dades tant d'aquest bus com de la memòria RAM (un bus extern anomenat MEM[11..0]). La dada que arriba a l'entrada de dades del MDR és seleccionada mitjançant un multiplexor que permet fer la selecció de la dada bé des del BUS 2 o bé des del bus extern MEM (vegeu la Fig. 7.5, Multiplexor a l'entrada del MDR).

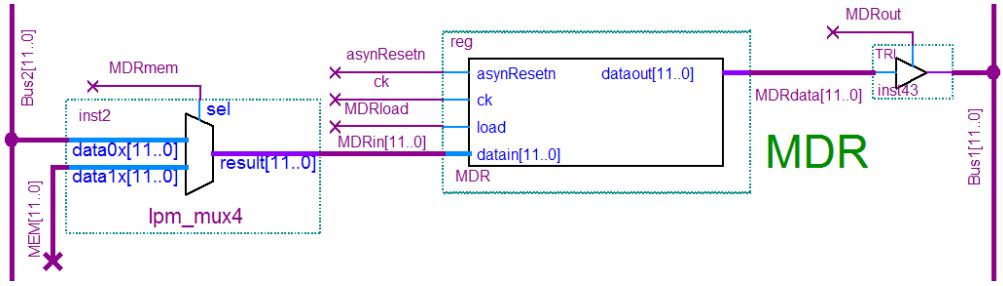


Fig. 7.5 Multiplexor a l'entrada del MDR

7.5.1.3. Registres

R1 i R2

Descripció
Són dos registres d'ús general que s'han d' implementar amb la cel·la reg.

Senyals connectats a les entrades

asynResetn: senyal de rèset asíncron, actiu a baixa.
ck: senyal de rellotge amb flanc actiu de pujada.
R1load/R2load: senyal de càrrega del registre.
Dades: les dades vénen del Bus 2.

Senyals connectats a les sortides

R1data [11..0] i R2data [11..0]: estan connectats a sengles buffers tristate múltiples que controlen el pas cap al Bus 1.

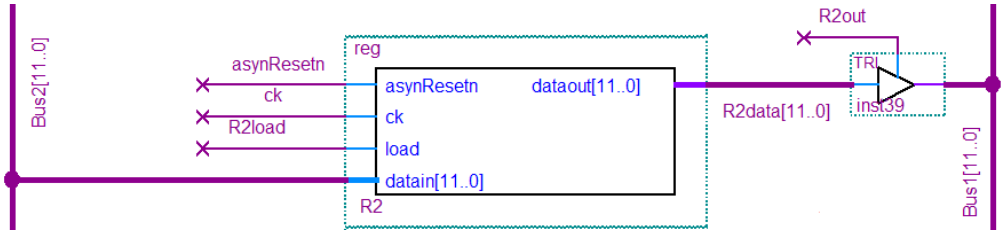


Fig. 7.6 Exemple del registre R2

IR (*Instruction Register*)

Descripció
Registre encarregat d'emmagatzemar la instrucció que ha de ser executada a continuació. Té la mateixa estructura de cel·la que els registres R1 i R2. S'implementa amb la cel·la reg.

Senyals connectats a les entrades
asynResetn: senyal de rèset asíncron, actiu a baixa.
ck: senyal de rellotge amb flanc actiu de pujada.
IRload: senyal de càrrega del registre.
Les dades vénen del Bus 2.

Senyals connectats a les sortides
IRdata [11..0] connectat a un buffer tristate múltiple que controla el pas cap al Bus 1. Els quatre primers bits IRdata[11..8] constitueixen una sortida de la UP.

PC (Program Counter)

Registre encarregat d'emmagatzemar l'adreça física de la següent instrucció que ha de ser executada. L'adreça emmagatzemada correspon a una posició de memòria de la RAM. Aquesta cel·la ha estat creada prèviament mitjançant la captura esquemàtica.

Senyals connectats a les entrades
asynResetn: senyal de rèset asíncron, actiu a baixa.
ck: senyal de rellotge amb flanc actiu de pujada.
PCinit: senyal d'inicialització del registre.
PCload: senyal de càrrega del registre.
Les dades vénen del Bus 2.

Senyals connectats a les sortides
PCdata [11..0] connectat a un buffer tristate múltiple que controla el pas cap al Bus 1.

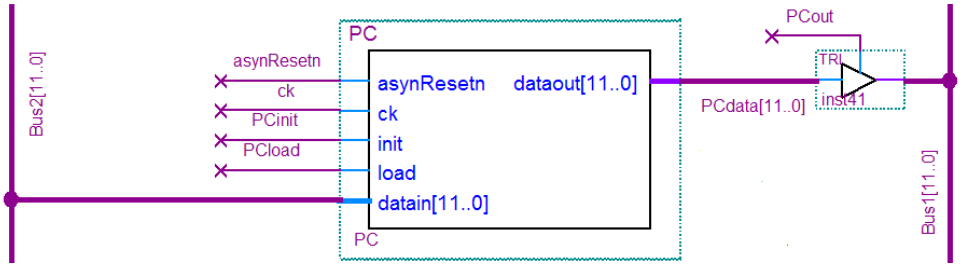


Fig. 7.7 Exemple del registre PC

MAR (*Memory Address Register*)

Descripció
Registre encarregat d'emmagatzemar l'adreça de la posició de la memòria RAM on es farà una operació de lectura o escriptura. Té la mateixa estructura de cel·la que els registres R1 i R2. S'implementa amb la cel·la reg.

Senyals connectats a les entrades
asynResetn: senyal de rèset asíncron, actiu a baixa.
ck: senyal de rellotge amb flanc actiu de pujada.
MARload: senyal de càrrega del registre.
Les dades vénen del Bus 2.

Senyals connectats a les sortides
MARdata [11..0] que està connectat a una sortida de la UP (camí directe a la RAM) i a un buffer tristate múltiple que controla el pas cap al Bus 1.

MDR (*Memory Data Register*)

Descripció
Registre encarregat d'emmagatzemar una dada relacionada amb la RAM per una operació de lectura o escriptura. Té la mateixa estructura de cel·la que els registres R1 i R2. S'implementa amb la cel·la reg.

Senyals connectats a les entrades
asynResetn: senyal de rèset asíncron, actiu a baixa.
ck: senyal de rellotge amb flanc actiu de pujada.
MDRload: senyal de càrrega del registre.
Les dades vénen de la sortida d'un multiplexor.

Senyals connectats a les sortides
MDRdata [11..0] connectat a una sortida de la UP (camí directe a la RAM) i a un buffer tristate múltiple que controla el pas cap al Bus 1.

Comentaris addicionals
Les dades d'entrada a aquest registre (MDRin [11..0]) provenen d'un multiplexor amb les especificacions següents:

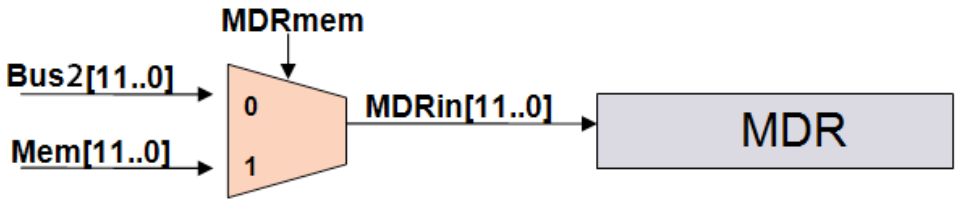


Fig. 7.8 Multiplexació de les dades d'entrada del MDR

Creació

Per a la creació d'aquest multiplexor hem d'anar a *Tools/MegaWizard Plug- In Manager...* i tot seguit hem de fer *Next* per crear una nova megafunció.

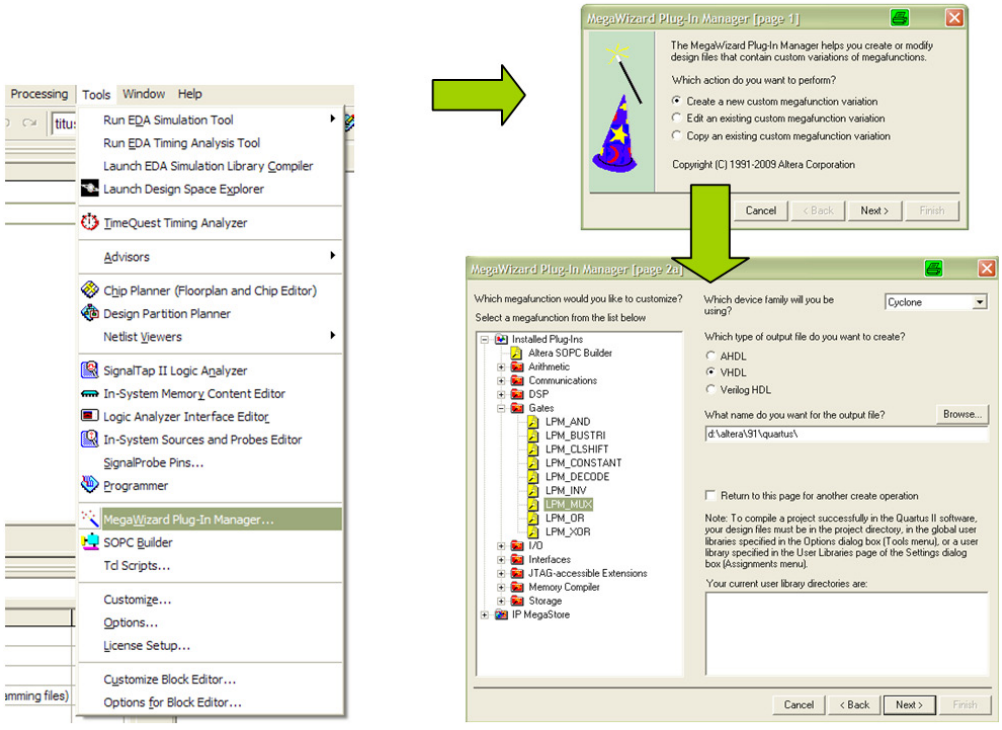


Fig. 7.9 Procés de creació d'un multiplexor de N bits d'entrada i M canals

Tot seguit, despleguem les opcions de *Gates*, i escollim *LPM_MUX*. Seleccionem l'opció VHDL perquè creï tot el codi amb aquest llenguatge. Li posem el nom de *lpm_mux4* dins del nostre directori de treball, i fem *Next*. Ara, a la finestra següent, li diem que volem dues entrades de 12 bits cada una d'elles, i premem el botó *Next* dues vegades. En aquesta darrera finestra seleccionem l'opció que falta (*lpm_mux4.bsf*) i premem el botó *Finish*.

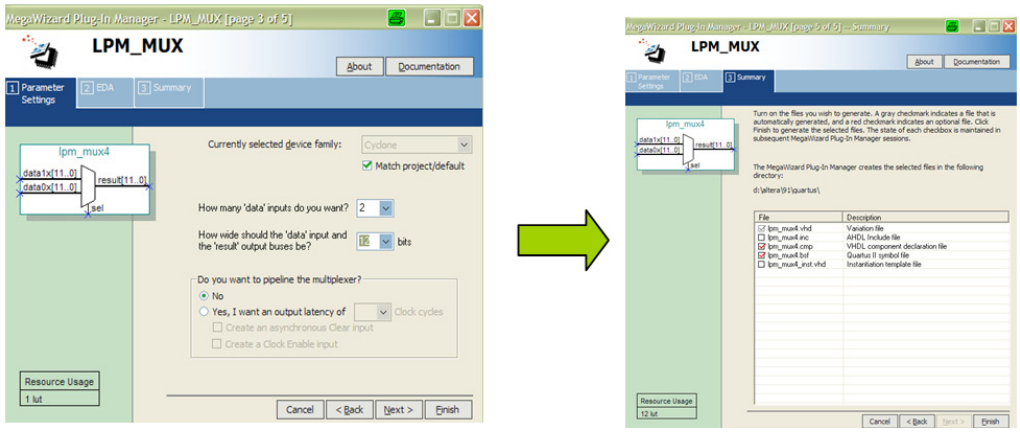


Fig. 7.10 Part final del procés de creació del multiplexor del MDR

Si ara fem doble clic sobre la nostra àrea de disseny, desplegant la llibreria project, trobem la nostra cel·la lpm_mux4.

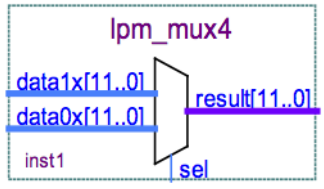


Fig. 7.11 Multiplexor del MDR

ACC (Acumulador)

Descripció

Registre encarregat d'emmagatzemar temporalment el primer operand de l'ALU a les operacions binàries. Rep les dades del Bus 1. S'implementa amb la cel·la reg.

Senyals connectats a les entrades

asynResetn: senyal de rèset asíncron, actiu a baixa.
ck: senyal de rellotge amb flanc actiu de pujada.
ACCload: senyal de càrrega del registre.
Les dades vénen del Bus 1.

Senyals connectats a les sortides

ACCdata [11..0] connectat a l'entrada A[11..0] de l'ALU.

7.5.1.4. ALU

Descripció
La unitat aritmeticològica és l'encarregada de realitzar totes les operacions de processament de dades del processador.

Entrades

A[11..0] que es connecta a la sortida de l'acumulador, on aquest bus l'anomenem ACCdata [11..0].
B[11..0] que es connecta directament al Bus1[11..0].
sel[2..0] que es connecta al pin_d'entrada de la UP ALUcode [2..0] que permet seleccionar l'operació de l'ALU.
ck: entrada de rellotge per sincronitzar les dades; va connectada al pin ck.

Sortides

C[11..0] bus de sortida de 12 bits connectat al Bus2[11..0].
carry, zero i **negative** es connecten a pins de sortida de la UP amb el mateix nom.

Operacions de l'ALU:
— *Pas transparent (C ← B)*
— *Resta (C ← A - B)*
— *Suma (C ← A + B)*
— *Màscara dels quatre primers bits (C ← B and 0FF)*
— *Incrementador d'una unitat (C ← B+1)*
— *Rotació a la dreta amb inclusió de carry*
— *Desplaçament a la dreta amb inclusió de carry*

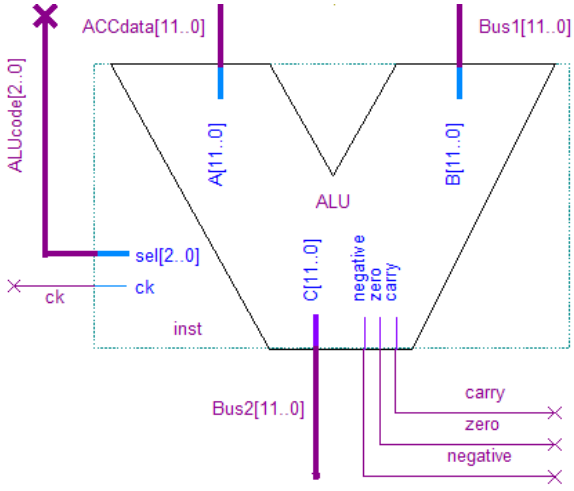


Fig. 7.12 Unitat Aritmèticològica

7.5.2. Arquitectura genèrica de la UP

Així doncs, una vegada donada la relació de registres, busos i elements interns, la UP hauria de quedar amb un aspecte semblant al disseny que es mostra a la Fig. 7.13.

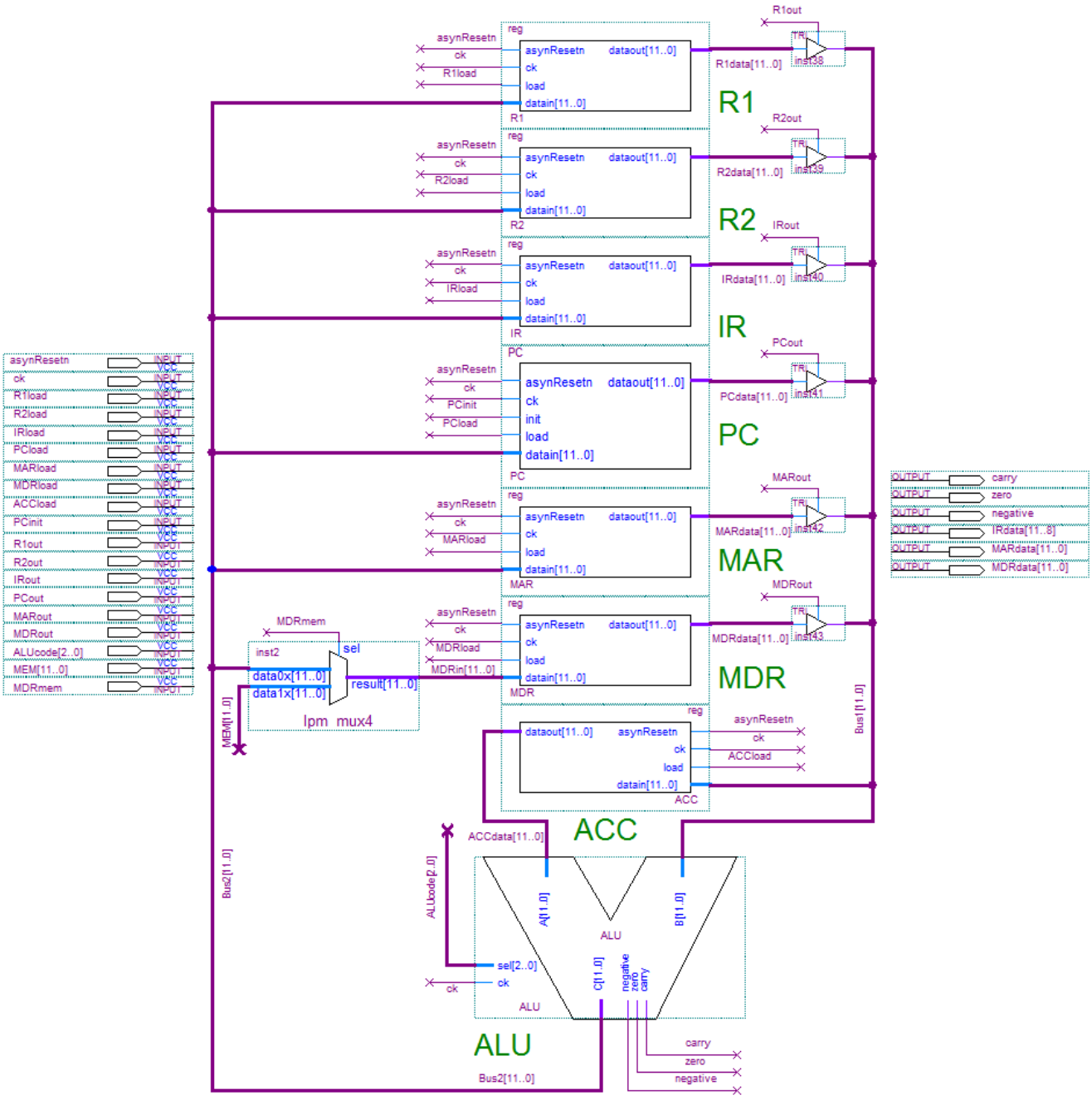


Fig. 7.13 Esquema intern de la Unitat de procés

Entrades

asynResetn: bit que fa un rèset de tots el registres.

ck: senyal de rellotge de tots els registres i l'ALU.

R1load, R2load, IRload, PCload, MARload, MDRload, ACCload: són els senyals de càrrega dels registres corresponents. Quan un d'aquests pins esdevé actiu, el registre corresponent carrega el valor de la seva entrada datain[11..0].

PCinit: bit que inicialitza el PC amb un valor predeterminat (p. ex., el valor d'1).

R1out, R2out, IRout, PCout, MARout, MDRout: permet fer que el valor emmagatzemat al registre corresponent pugui ser transmès al bus. Aquests bits controlen els senyals de pas dels *buffer tristate*.

ALUcode[2..0]: permet indicar a l'ALU l'operació a realitzar.
MEM[11..0]: dada de la memòria RAM connectada al canal corresponent del multiplexor del MDR; aquesta dada és el bus de dades de sortida de la RAM.
MDRmem: bit que permet indicar al multiplexor del MDR quina dada ha de carregar (MEM[11..0] o Bus2[11..0]).

Sortides

carry: bit corresponent a la sortida carry de l'ALU.
zero: bit corresponent a la sortida zero de l'ALU.
negative: bit corresponent a la sortida negative de l'ALU.
IRdata[11..8]: són els 4 bits de més pes de la sortida del IR (no de la sortida al Bus 1); aquesta dada és *l'operation code* (o codi d'instrucció).
MARdata[11..0]: són els 12 bits de la sortida de la MAR (no de la sortida al Bus 1); aquesta dada és el bus d'adreces de la RAM.
MDRdata[11..0]: són els 12 bits de la sortida de la MDR (no de la sortida al Bus 1).

7.6. Simulació de la UP

Una vegada dissenyada la UP, heu de simular-la per comprovar que el seu comportament és correcte. Us recomanem que prepareu una simulació amb retards d'acord amb les indicacions següents:

- Assegureu-vos que totes les cel·les corresponents als registres i a l'ALU tenen assignat un nom d'instància, com es mostra a la Fig. 7.13. Si heu de fer canvis al circuit, feu-los, deseu el circuit i recompileu-lo.

- Creeu un nou fitxer d'ones per especificar els vectors de simulació (**`File`** → **`New`** → **`Verification/Debugging Files`** → **`Vector Waveform File`**).

- Establiu 3 microsegons com a duració de la simulació (**`Edit`** → **`End Time`** 3us)

- Per facilitar el dibuix de les formes d'ona, establiu un engraellat de 50 ns (**`Edit`** → **`Grid Size`** 50 ns).

- A continuació hi ha una llista amb els senyals que heu d'afegir a l'editor de formes d'ona. Conté totes les entrades, totes les sortides i els nodes interns més importants. Al capítol 4 (secció 4.3.2) s'explicava com se seleccionen les entrades i com s'afegeixen a l'editor. A la finestra del **`Node Finder`**, per veure solament les entrades s'havia d'especificar al camp **`Filter`** l'opció **`Pins: Input`**. De la mateixa manera, si voleu veure solament les sortides heu d'especificar l'opció **`Pins: Output`**. Finalment, per accedir als nodes interns heu d'especificar l'opció **`Post-Compilation`** (vegeu Fig. 7.14). La llista de senyals és la següent:

Entrades

ACCload, ALUcode (bus), asynResetn, ck, IRload, IRout, MARload, MARout, MDRload, MDRmem, MDRout, MEM (bus), PCinit, PCload, PCout, R1load, R1out, R2load, R2out

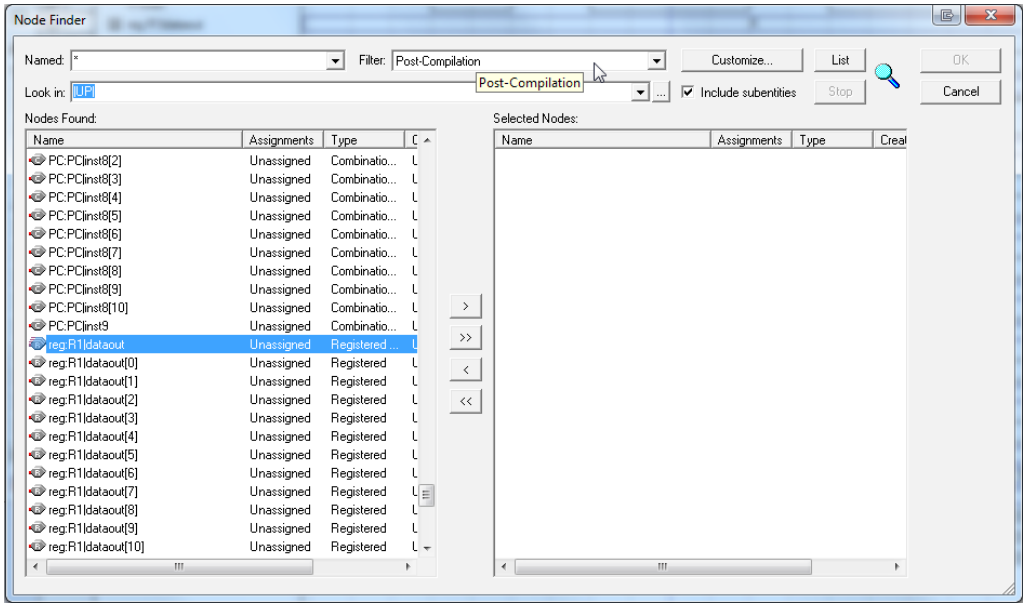


Fig. 7.14 Selecció dels nodes interns

Sortides

carry, IRdata (bus), MARdata (bus), MDRdata (bus), negative, zero

Nodes interns

ALU:ALU|C	*bus de 12 bits ([11..0]); sortida C de l'ALU*
reg:ACC|dataout	*bus de 12 bits ([11..0]); contingut d'ACC*
reg:IR|dataout	*bus de 12 bits ([11..0]); contingut d'IR*
reg:R1|dataout	*bus de 12 bits ([11..0]); contingut de R1*
reg:R2|dataout	*bus de 12 bits ([11..0]); contingut de R2*
PCdataout	*bus de 12 bits ([11..0]); contingut de PC (vegeu el paràgraf següent)*

El senyal PCdataout no el trobareu amb aquest nom. L'haureu de «construir» fent els passos següents:

- A la finestra del **`Node Finder`**, seleccioneu els dotze senyals relatius a les instàncies dels *flip-flops* de la cel·la PC (tenen el nom PC:PC|ff1[x] i PC:PC|ff0[x] si vau respectar els noms d'instàncies que es recomanaven a la fi de la secció 5.9.3)
- Afegiu els senyals a l'editor d'ones
- Ordeneu-los del bit 11 al bit 0 de PC. Per ordenar els senyals, els podeu moure a la posició desitjada arrossegant-los amb el ratolí.
- Seleccioneu els dotze senyals i definiu un grup de senyals assignant-li el nom **PCdataout**. Per definir un grup de senyals, una vegada els senyals estan seleccionats, heu de fer clic amb el botó dret del ratolí i executar l'ordre **`Grouping`** → **`Group...`**. Al formulari que es presenta introduïu el nom del grup, és a dir, **PCdataout**.

• Configureu el format de visualització dels busos. Per fer-ho, a l'editor d'ones posicioneu el ratolí sobre el bus la configuració del qual voleu establir, feu clic amb el botó dret i executeu **`Properties`** → **`Radix`** → ***`Format desitjat`***. Establiu el format de visualització dels busos de la manera següent:

- l'entrada MEM, les sortides MARdata i MDRdata, i els busos interns ALU:ALU|C, reg:ACC|dataout, reg:R1|dataout i reg:R2|dataout en format decimal signat
- la sortida MARdata, i els nodes interns reg:IR|dataout i PCdataout en format decimal no signat.
- l'entrada ALUcode i la sortida IRdata en format binari.

• A continuació hi ha el guió de la simulació. El rellotge **ck** inicialment val 0 i té un període de 100 ns amb un *duty cicle* del 50%. Cada línia del guió requereix 100 ns i s'implementa mitjançant l'activació dels corresponents senyals d'entrada de la UP. Els senyals d'entrada canvien sincronitzats amb el flanc de baixada del rellotge i es mantenen actius un període de rellotge. A la Fig. 7.15 podeu veure com s'implementen les quatre primeres línies del guió. No heu de buscar un sentit global al que es fa durant aquesta simulació: simplement es mouen dades entre els diversos elements de la UP per comprovar el seu correcte funcionament.

- *Reset* asíncron de tots els registres
- Inicialització síncrona del PC al valor constant predeterminat
- MDR ← Valor 45 des de MEM
- R1 ← MDR
- R2 ← Increment(MDR)
- ACC ← R1
- R1 ← ACC + R2
- PC, ACC ← R1
- R1 ← ACC - R2
- ACC ← R2
- R1 ← ACC - R2
- MDR ← Valor 1145 des de MEM
- IR ← MDR
- MAR ← Màscara(IR)
- R1 ← Rotació(MAR)
- R1 ← Increment(IR)
- R1 ← Rotació(MAR)
- R1 ← Increment(IR)
- PC ← Increment(PC)
- R1 ← Desplaçament(MAR)
- ACC ← R2
- R2 ← ACC - R2
- R1 ← Desplaçament(MAR)
- ACC ← R2
- R2 ← Increment(R2)
- R1 ← ACC - R2
- R1 ← Increment(R1)
- MDR ← R2
- *Reset* asíncron de tots els registres

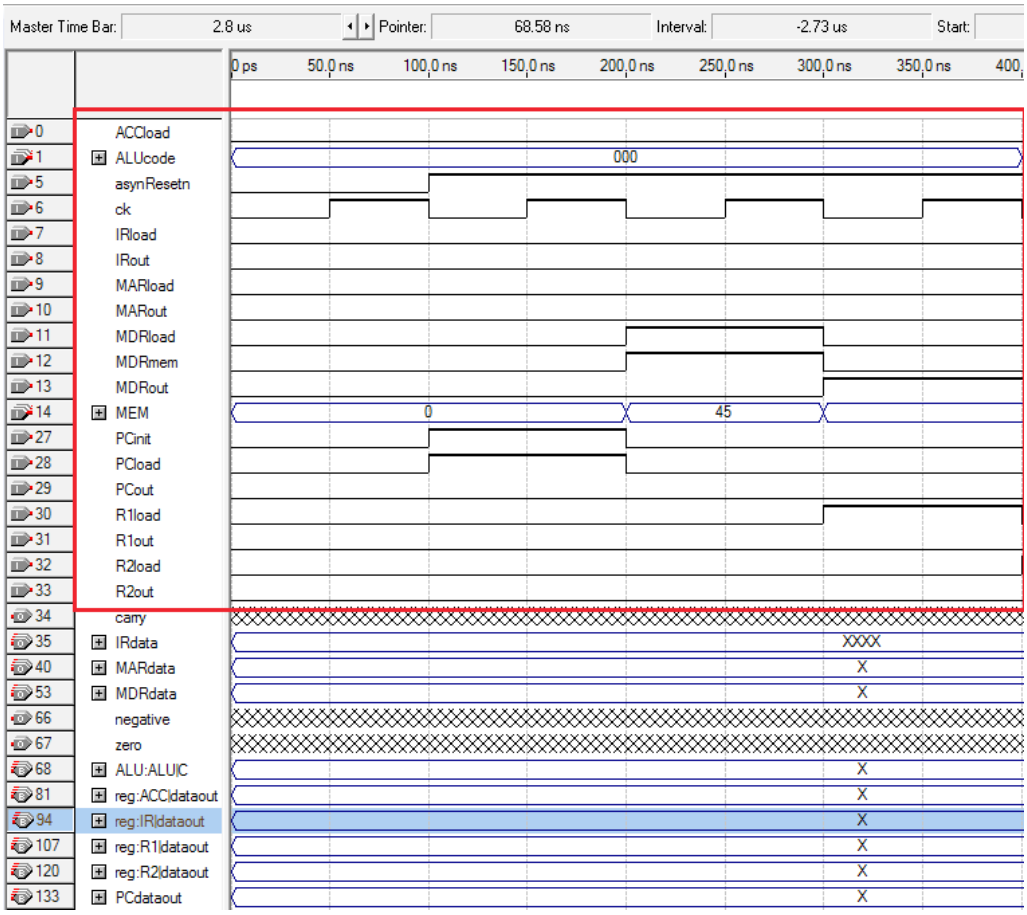


Fig. 7.15 Implementació de les quatre primeres línies del guió de la simulació

- Per facilitar la llegibilitat dels resultats de la simulació, ordeneu els senyals d'acord amb la llista següent. Recordeu que per ordenar els senyals, els heu de moure a la posició desitjada arrossegant-los amb el ratolí. A la Fig. 7.16 podeu veure els senyals ja ordenats a l'editor d'ones.

asynResetn	*entrada*
ck	*entrada*
R1load	*entrada*
reg:R1|dataout	*node intern, bus de 12 bits ([11..0]); contingut de R1*
R1out	*entrada*
R2load	*entrada*
reg:R2|dataout	*node intern, bus de 12 bits ([11..0]); contingut de R2*
R2out	*entrada*
IRload	*entrada*
reg:IR|dataout	*node intern, bus de 12 bits ([11..0]); contingut d'IR*
IRdata	*sortida, bus de 4 bits ([11..8]); 4 primers bits d'IR*
IRout	*entrada*
PCinit	*entrada*
PCload	*entrada*
PCdataout	*node intern, bus de 12 bits ([11..0]); contingut de* PC
PCout	*entrada*

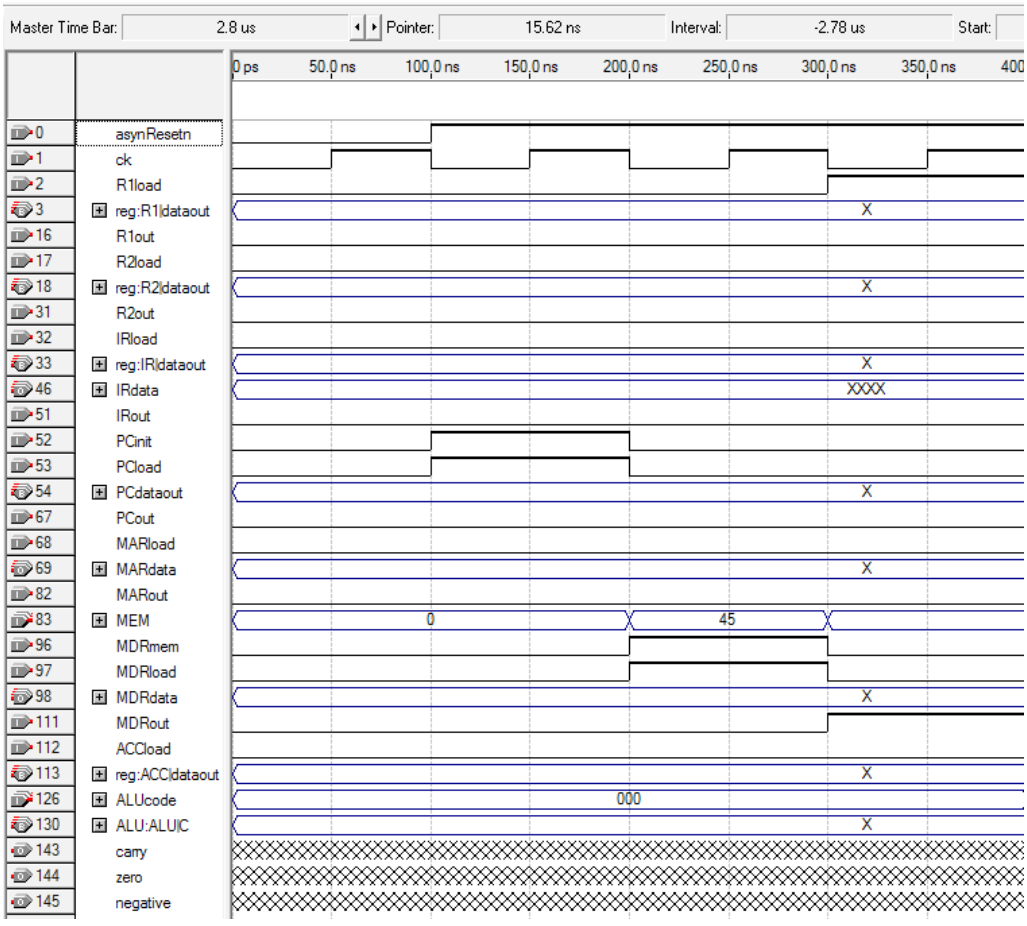


Fig. 7.16 Preparació de la simulació: senyals ja ordenats

MARload	*entrada*
MARdata	*sortida, bus de 12 bits ([11..0]); contingut de MAR*
MARout	*entrada*
MEM	*entrada, bus de 12 bits ([11..0]); dades de la memòria*
MDRmem	*entrada*
MDRload	*entrada*
MDRdata	*sortida, bus de 12 bits ([11..0]); contingut de MDR*
MDRout	*entrada*
ACCload	*entrada*
reg:ACC|dataout	*node intern, bus de 12 bits ([11..0]); contingut d'ACC*
ALUcode	*entrada, bus de 3 bits ([2..0]);*
ALU:ALU|C	*node intern, bus de 12 bits ([11..0]); sortida C de l'ALU*
carry	*sortida*
zero	*sortida*
negative	*sortida*

- Graveu el fitxer d'ones amb el nom **UP.vwf**. Recordeu que heu d'especificar aquest fitxer en la configuració de la simulació (**`Assignments`** → **`Settings`** → **`Simulator Settings`** → **`Simulation input`**).

- Ara ja podeu llençar la simulació.

- Durant el procés de verificació dels resultats de la simulació, quan aneu a comprovar la correcta realització de les operacions de emmascarament, de rotació i de desplaçament, és recomanable que canvieu momentàniament el format de visualització dels busos involucrats a format binari. D'aquesta manera us resultarà més senzill de veure si aquestes operacions es fan bé. Per exemple, quan comproveu l'operació «R1 ← Rotació(MAR)» seria convenient que configuréssiu en format binari els busos MARdata, ALU:ALU|C i reg:R1|dataout.

8. Disseny de la unitat de control

Com ja s'ha vist anteriorment, la unitat de control és l'encarregada de generar la seqüència d'ordres necessàries per executar cada instrucció emmagatzemada a la RAM.

Aquesta RAM està formada per un conjunt de posicions de memòria on cada posició correspon a una instrucció de 12 bits on podríem diferenciar dues parts de la mateixa instrucció:

- Part alta de la instrucció [11..8] també anomenada *opcode*. Aquests 4 bits són els encarregats de codificar la instrucció a executar. D'aquesta manera podem tenir fins a un total de 16 instruccions o *instruction set*. Aquest codi d'instrucció no correspon a l'adreça interna de la ROM on es troba el seu microcodi; prèviament a l'execució d'aquesta instrucció cal descodificar-la per tal d'identificar on es troba realment el microcodi específic vinculat a aquest *opcode*.
- Part baixa de la instrucció [7..0] anomenada operand. Aquests 8 bits poden tenir diferents significats segons l'*opcode* que l'acompanya:
  - Dada numèrica de 8 bits, p. e.: *0x4A*.
  - Adreça de 8 bits on es troba la dada a processar dins de la RAM, com si es tractés d'un punter.
  - Adreça de salt en cas d'una instrucció de salt condicional o incondicional.
  - Res, cas en el qual aquesta regió de la instrucció estaria a *0x00*.

D'aquesta manera cada instrucció d'alt nivell (*opcode*) s'implementa mitjançant un conjunt de microinstruccions emmagatzemades a una ROM de microcodi. Aquestes microinstruccions contenen tots els senyals de control de la UP necessaris per moure el flux de les dades dins la UP i així poder executar la instrucció desitjada.

Per tal de generar aquest seguit d'ordres és necessària la incorporació d'una sèrie d'unitats específiques per processar cadascuna de les instruccions a executar i traduir-les en un seguit de microinstruccions (microcodi), que és l'encarregat d'ordenar a la unitat de procés la seva execució correcta.

Per fer el disseny de la UC el més semblant possible a la realitat s'ha decidit desenvolupar-la mitjançant un seqüenciador. Aquest permet determinar la direcció de la microinstrucció següent en funció d'uns paràmetres determinats pel seu entorn.

8.1. Procés a dur a terme per a l'execució d'una instrucció

Per poder executar correctament una instrucció és necessari dur a terme un seguit de tasques que a continuació comentem breument:

- Cicle de *fetch* o cerca de la instrucció. Aquest cicle està format per un total de 5 microinstruccions que tenen com a objectiu anar a buscar la instrucció a executar a la memòria RAM i portar-la a l'IR. A més, el PC s'incrementa.
- Cicle d'execució de la instrucció. Té com a objectiu l'execució pròpiament dita de la instrucció. Cada instrucció implementa aquest cicle mitjançant un conjunt concret de microinstruccions.

8.2. Elements bàsics de la unitat de control

Per poder dur a terme els processos anteriors és necessari el disseny d'un conjunt de mòduls. D'aquesta manera hem de dissenyar:

- Un seqüenciador format alhora per:
  - Un incrementador
  - Un multiplexor de 4 canals
  - Un registre (microPC)
- Una ROM per emmagatzemar els microprogrames
- Un multiplexor
- Una lògica de control del seqüenciador
- Un descodificador d'instruccions
- Una xarxa de busos
- Lògica combinacional senzilla

Al gràfic següent podem observar l'estructura interna de la unitat de control.

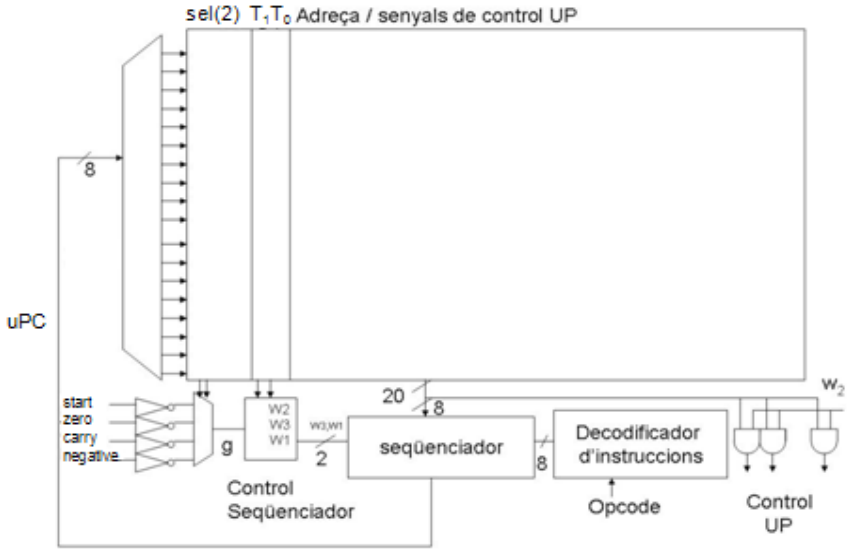


Fig. 8.1 Esquema intern de la Unitat de control

8.3. Descripció de la funcionalitat de cada element

8.3.1. Seqüenciador (sequencer)

Descripció

L'objectiu del seqüenciador és entregar a la seva sortida l'adreça de la microinstrucció següent a executar.

Està bàsicament format per tres elements gairebé ja dissenyats.

Té tres modes de funcionament en funció dels senyals W1 i W3:

— Mode d'*incrementació* del microPC. En aquest mode el microPC s'incrementa en '1'. Permet generar la microinstrucció següent a executar a partir de la microinstrucció anterior; pensat per a les microinstruccions de tipus EXE i IF amb condició no complerta.

— Mode de càrrega de la *input_address[7..0]* que permet carregar al microPC una adreça de salt emmagatzemada a la ROM de microcodi (pensat per a les microinstruccions de tipus GOTO i IF amb condició complerta).
— Mode de càrrega de la *decodinstruct[7..0]* que permet carregar al microPC una adreça de salt externa (adreça d'inici del cicle d'execució). Està pensat per indicar l'adreça on es troba la primera microinstrucció del cicle d'execució de la instrucció que s'està executant en aquell moment. Quan es vol executar una instrucció en codi màquina emmagatzemada a l'IR, hem d'indicar al sistema on està ubicat l'inici (adreça) del cicle d'execució d'aquesta instrucció dins del microcodi emmagatzemat a la ROM; és a través d'aquest mode de càrrega que s'introdueix al microPC aquesta adreça d'inici.

Entrades

asynResetn: senyal de *rèset* asíncron, actiu a baixa.
ck: senyal de rellotge amb flanc actiu de pujada.
W3, W1: bits de control del multiplexor de dades.
input_address [7..0]: dades que provenen de la ROM de microinstruccions (senyal control_signals[19..12]) i que poden constituir una adreça de salt.
decodInstruc [7..0]: dades que provenen del descodificador d'instruccions a través d'un bus intermedi anomenat amb el mateix nom i que constitueixen l'adreça de salt a l'inici del cicle d'execució.

Sortides

output_address [7..0]: es connecta a un bus anomenat address[7..0] que va a parar al bus d'adreces de la ROM. Per aquesta sortida surt l'adreça de la microinstrucció següent a executar.

Informació complementària

Aquest element està format per tres blocs: un MUX4*1, un incrementador i un registre de 8 bits (també anomenat microPC). L'estructura interna seria, a grans trets, la següent:

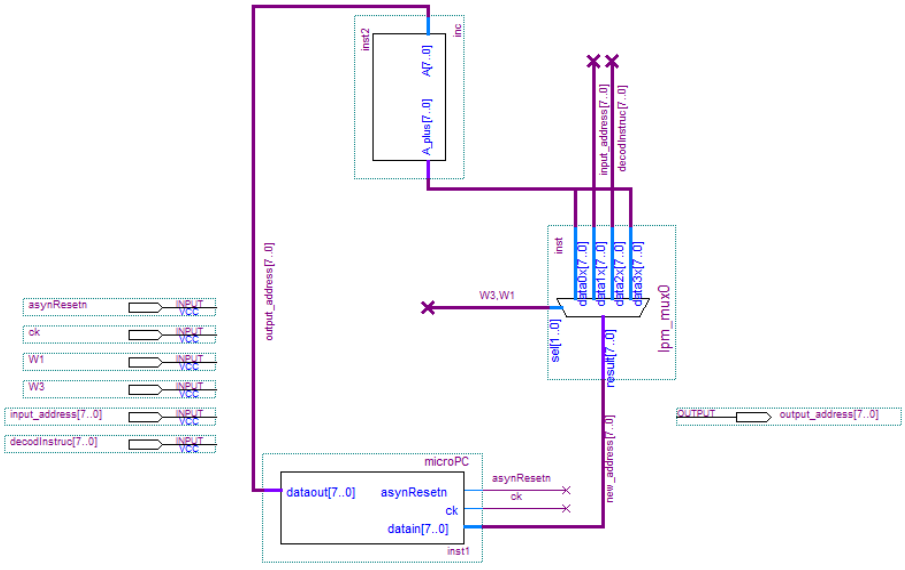


Fig. 8.2 Esquema intern del seqüenciador

8.3.1.1. Multiplexor

Descripció

Multiplexor amb 4 canals d'entrada de 8 bits cada un, amb dos bit de selecció, W3 (de més pes) i W1 (de menys pes).

Per a la creació d'aquest multiplexor s'ha d'anar a *Tools/MegaWizard Plug- In Manager...* i tot seguit fem el procés vist a l'apartat del MDR (pàg. 97).

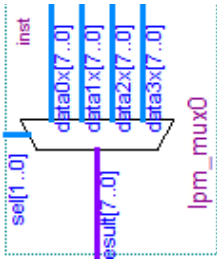


Fig. 8.3 Multiplexor d'entrades del seqüenciador

Senyals connectats a les entrades

Canal 0 i 3: **A_plus[7..0]** (sortida de 8 bits de l'incrementador)
Canal 1: **input_address [7..0]** (pin d'entrada)
Canal 2: **decodInstruc [7..0]** (pin d'entrada)

Senyal connectat a la sortida

La sortida es connecta a l'entrada datain[7..0] del microPC, a través del bus intermedi **new_address[7..0]**

8.3.1.2. microPC

Descripció

Es tracta d'un registre de 8 bits que té les entrades i sortides següents.

Entrades

asynResetn: senyal de *rèset* asíncron, actiu a baixa.
ck: senyal de rellotge amb flanc actiu de pujada.
datain[7..0]: bus de 8 bits que es connecta a la sortida del multiplexor a través del bus intermedi new_address[7..0].

Sortides

dataout[7..0]: bus de 8 bits que es connecta al pin de sortida del seqüenciador output_address[7..0] i al mateix temps al bus d'entrada de l'incrementador.

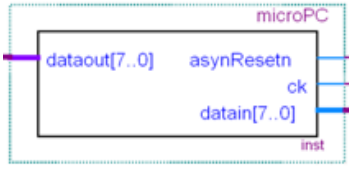


Fig. 8.4 microPC (imatge girada: entrades a la dreta)

8.3.1.3. Incrementador (inc)

Descripció
Dissenyat en VHDL. Es tracta d'una unitat encarregada d'incrementar en 1 el valor introduït a la seva entrada.

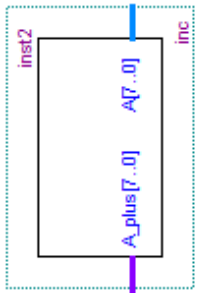


Fig. 8.5 L'incrementador

Entrades
A[7..0]: A aquesta entrada li connectem la sortida del microPC.

Sortides
A_plus[7..0]: Aquesta sortida va connectada a un canal del multiplexor prèviament especificat.

8.3.2. Descodificador d'instruccions (IRdecoder)

Descripció
El processador que volem desenvolupar ha d'estar capacitat per executar diferents instruccions emmagatzemades a la memòria RAM.

A causa del fet que el nostre processador té un número d'instruccions reduït i que el nostre *opcode* tan sols és de 4 bits cal descodificar aquest *opcode* i convertir-lo a una adreça real de memòria per tal d'identificar adequadament on comença els seu cicle d'execució dins de la ROM.

Està clar que una instrucció qualsevol (*opcode*) per poques línies de microcodi que ocupi a la ROM està format per un mínim de 3 microinstruccions (més endavant ho justificarem); això vol dir que si tinguéssim 16 instruccions d'alt nivell (*opcode*) a una mitjana de 4 microinstruccions per *Opcode* + el cicle de cerca de la instrucció (cicle de *fetch*) que pot ser de 6 microinstruccions més, podríem tenir aproximadament unes 70 línies de microcodi. Com veiem 70 línies no es poden codificar amb 4 bits sinó que en caldrien més aviat 7 bits *d'opcode*.

Això vol dir que si el programa sencer emmagatzemat en RAM ha d'estar format per un seguit *d'opcodes* (on aquests han de referenciar directament l'adreça interna de la ROM) i operands, necessitem una memòria amb posicions de 15 bits (7+8), amb l'increment consegüent d'espai requerit.

És a causa d'això que utilitzem un descodificador on donat un *opcode* de 4 bits obtenim l'adreça real de 8 bits on es troba la primera microinstrucció del cicle d'execució. Aquest descodificador va ser realitzat en VHDL anteriorment.

Bus d'entrada

opcode[3..0]. Es tracta d'un bus de 4 bits que es connecta al pin d'entrada de la UC del mateix nom.

Bus de Sortida

initAddress [7..0]. Bus de 8 bits connectat a un bus intermedi anomenat decodInstruc [7..0] que va directament connectat a l'entrada del seqüenciador amb el mateix nom.

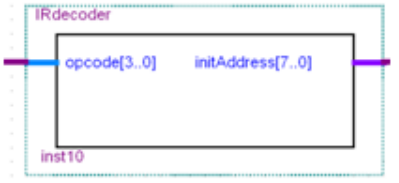


Fig. 8.6 IRdecoder

8.3.3. Multiplexor de condicions d'entrada

Descripció

Multiplexor amb 4 canals d'entrada de 1 bit cada un, amb dos bits de selecció, anomenats sel.

Per a la creació d'aquest multiplexor hem d'anar a *Tools/MegaWizard Plug- In Manager...* i tot seguit fem el procés vist a l'apartat del MDR.

L'objectiu d'aquest multiplexor és deixar passar un bit de condició d'entrada per tal d'avaluar-lo i determinar l'execució d'una microinstrucció o una altra (salt condicional). Es poden avaluar 4 bits:

Bit de start. Quan aquest bit estigui actiu es pot començar a executar tot el programa de la RAM.

Bit de zero (que prové de la UP i val '1' quan el resultat de l'operació anterior és igual a 0). Aquest bit es pot fer servir per a operacions de salt condicional (comparació de registres, salt i altres instruccions...).

Bit de carry. Bit que prové de la UP i val '1' quan el resultat d'una operació requereix un tretzè bit.

Bit de negative. Bit que prové de la UP i val '1' quan el resultat d'una operació és negatiu.

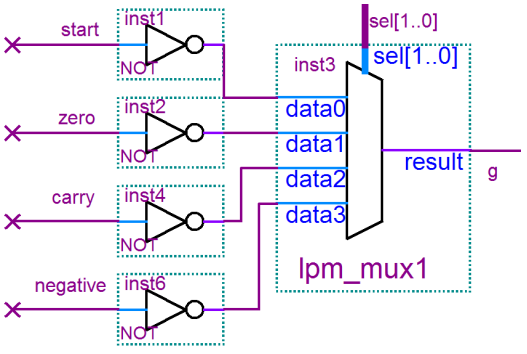


Fig. 8.7 Multiplexor de condicions d'entrada

Entrades

Canal 0: start negat (1 bit), és a dir, el pin d'entrada de la UC anomenat start ha d'anar connectat a aquest pin, però negat.

Canal 1: zero negat (1 bit). El pin d'entrada de la UC anomenat zero ha d'anar connectat a aquest pin, però negat.

Canal 2: carry negat (1 bit). El pin d'entrada de la UC anomenat carry ha d'anar connectat a aquest pin, però negat.

Canal 3: negative negat (1 bit). El pin d'entrada de la UC anomenat negative ha d'anar connectat a aquest pin, però negat.

sel[1..0]: connectat al control_signals [23..22] que és part del bus de sortida de la ROM que emmagatzema el microcodi a executar.

Sortides

g: connectat al bit d'entrada g de la unitat CtrlSeq.

8.3.4. ctrlSeq

Descripció

Aquesta cel·la té la finalitat de generar els senyals de control del seqüenciador.

Com ja hem comentat prèviament, el seqüenciador té l'objectiu de calcular la nova adreça de la microinstrucció que ha de ser executada. Aquesta adreça és una o una altra en funció del tipus de la microinstrucció que s'està executant: exe, if (condició), goto o bra.

A la taula següent resumim els tipus de microinstruccions amb els senyals de control que corresponen a cada una:

Tipus de microinstrucció	T1	T0		Sortida de ctrlSeq que s'activa
exe	1	0		W2
if	0	0	g=1	W1
goto	0	1		W1
bra	1	1		W3

Taula 8.1 Taula de control del seqüenciador
(g és una entrada de ctrlSeq indicadora de l'estat d'una determinada condició)

Exe

És la microinstrucció que permet executar un conjunt de microordres sobre la unitat de procés. Aquesta microinstrucció fa que el seqüenciador incrementi en 1 la posició del microPC i permet alhora que la dada emmagatzemada a la ROM (senyals de control dels registres de la UP i de l'ALU) es presenti cap a la UP a través d'una bateria de portes AND mitjançant l'activació del bit de control anomenat W2.

If

És una microinstrucció que depenent del valor del senyal de condició g, permetrà fer un salt a una adreça o anar a l'adreça següent.

Goto

Fa un salt incondicional a una adreça emmagatzemada a la posició actual de la ROM. Aquesta microinstrucció (junt amb l'if amb condició complerta) fa que el seqüenciador agafi l'adreça que li està entregant la ROM (entrada input_address de seqüenciador) i

carregui aquesta adreça al microPC. Aquestes dues microinstruccions (if i goto) controlen el bit W1.

Bra
Aquesta microinstrucció indica al seqüenciador que s'ha de fer un salt a l'inici del cicle d'execució de la instrucció actual. Aquesta adreça d'inici curiosament no prové de la ROM com en el cas de l'if o del goto sinó que prové de l'IRdecoder (el qual la genera a partir de *l'opcode* emmagatzemat a l'IR).

D'aquesta manera queden justificats els senyals que generen aquesta cel·la, i que són els següents:

$W1 = \overline{T1} \cdot \overline{T0} \cdot g + \overline{T1} \cdot T0$	Per executar l'if (amb condició certa) i el goto
$W2 = T1 \cdot \overline{T0}$	Per executar l'exe
$W3 = T1 \cdot T0$	Per executar el bra

Entrades

T1: connectat a control_signals [21].
T0: connectat a control_signals [20].
g: connectat a la sortida del multiplexor de condicions.

Sortides

W1: connectat al pin W1 de seqüenciador mitjançant el senyal anomenat W1.
W2: connectat a un senyal anomenat W2 i connectat a una bateria de 20 portes AND (o una porta AND amb una connexió de BUS de 20 pins a un dels seus nodes d'entrada) que faran de "porta de pas" dels senyals que surten de la ROM cap a la UP.
W3: connectat al pin W3 de seqüenciador mitjançant un senyal anomenat W3.

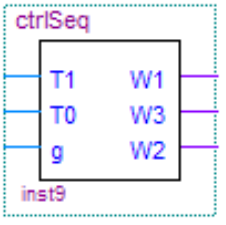


Fig. 8.8 Símbol de ctrlSeq

8.3.5. Bateria de Portes AND

Descripció
Comandades pel senyal W2 tenen com a objectiu permetre la càrrega dels senyals de la sortida de la ROM de microcodi sobre la unitat de procés. Aquest fet només esdevindrà possible quan s'estigui executant una microinstrucció de tipus exe.

Senyals connectats a les entrades

W2: a un dels pins de totes les portes.
control_signals [19..0]: Tots els bits que vénen de la ROM cap a la UP.

Senyals connectats a les sortides

control_signals_to_up [19..0]: Bits connectats al pin de sortida.

8.3.6. ROM de microcodi

Descripció
Aquesta unitat és l'encarregada d'emmagatzemar tot el microcodi equivalent a totes les instruccions del processador, així com el microcodi d'inicialització del processador i el de la cerca de la instrucció (*fetch*).

Per tal de dissenyar una ROM, de la mateixa manera que hem dissenyat abans els multiplexors, farem servir el mateix WIZARD (*Tools/MegaWizard Plug- In Manager...)* i tot seguit farem el procés vist a l'apartat 7.5.1.3 del MDR, però tenint en compte que ara el component es troba a la secció *Memory Compiler* i model *ROM:1-PORT* (vegeu les figures següents).

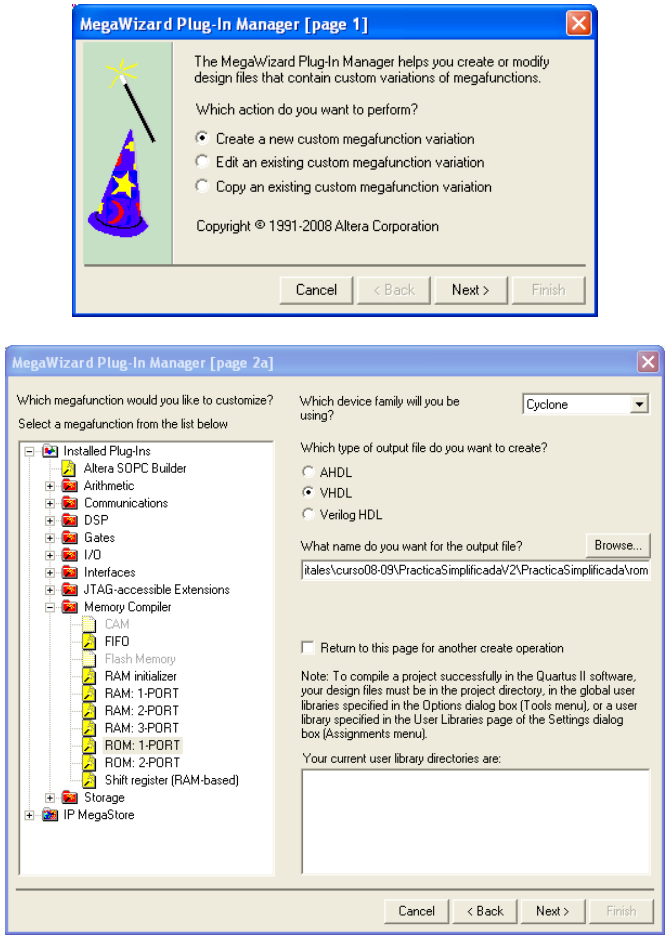


Fig. 8.9 *Wizard* de creació d'una ROM

Com s'ha anat veient en els apartats anteriors, aquesta ROM hauria de tenir les característiques següents:

— Tipus de família: Cyclone o Cyclone II (depenent de l'FPGA on s'implementarà).
— Tipus de fitxer de sortida: VHDL.
— Nom del fitxer de sortida: <directori del projecte>/rom.
— Amplada del bus de sortida (bus q): 24 bits, que són els bits que hem vist fins ara als apartats anteriors, per tal de controlar el multiplexor, el ctrlSeq, el seqüenciador i la UP. El nombre de posicions de memòria (capacitat de microinstruccions) ha de ser de 256; suficient pel nombre de microinstruccions que volem desenvolupar.
— El tipus de *block* de memòria ha de ser en mode Auto.
— Ha de tenir un sol rellotge (single clock).

Cap més opció no ha de quedar activada (desactiveu l'opció *'q' output port*), tot i que ens hem d'assegurar que el nom del fitxer que conté totes les dades del microcodi s'anomena romv1q.mif (s'ha d'introduir manualment a la pàgina 5/7 d'aquest *Wizard*).

A la pàgina 7 del *wizard*, es marca la casella relativa a la generació del símbol rom.bsf.

Finalment es respon afirmativament la pregunta de si es vol afegir aquest element al projecte.

Entrades

clock: ha d'estar connectat al pin *ck* de la UC però negat; així mentre la UP treballa per flanc de pujada, la UC ho fa per flanc de baixada.

address[7..0]: Connectat al bus address[7..0] que ve de la sortida del seqüenciador output_address[7..0].

Sortides

q[23..0]: Connectat al bus control_signals [23..0] que es reparteix de la manera següent:

— **control_signals [23..22]** pels bits de sel del multiplexor de condicions.
— **control_signals [21..20]** pel T1 i T0 del ctrlSeq.
— **control_signals [19..12]** pels 8 bits input_address[7..0] del seqüenciador.

La resta de senyals són microordres

Cadascun d'aquests senyals (control_signals[23..12]) els fem passar a través d'un buffer (cel·la LCELL) per facilitar posteriorment la visualizatció de senyals durant la simulació del processador. Com podreu veure a la secció 8.4, es fan servir tres grups d'instàncies de la cel·la LCELL (de 2, 2 i 8 instàncies, respectivament) i cadascun d'ells té el seu nom d'instància especificat (*_sel*, *_T* i *_jumpAddress*).

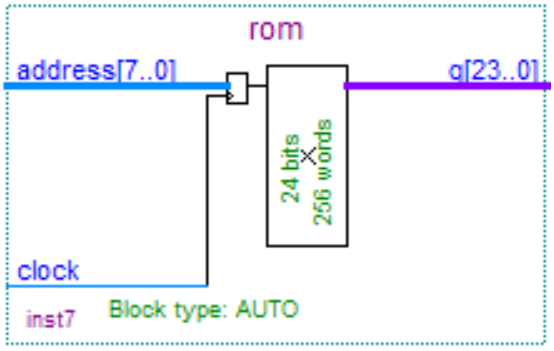


Fig. 8.10 Símbol de la ROM

A la figura següent podeu veure un cronograma que explica el funcionament de la ROM. Podeu observar que la lectura es realitza regida pel flanc de pujada del rellotge.

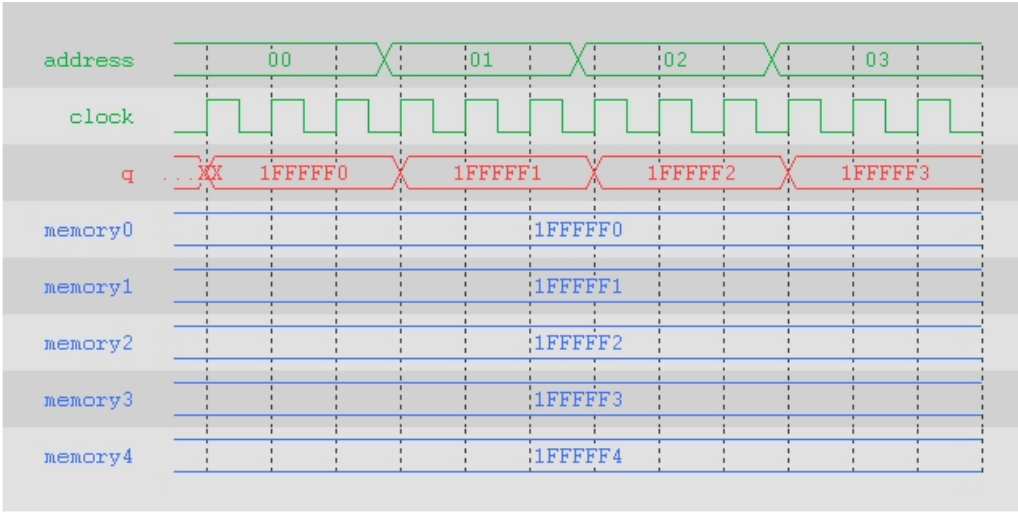


Fig. 8.11 Cronograma de sortida de la ROM

8.4. Arquitectura genèrica de la UC

D'aquesta manera, i donada la relació entre la ROM i els elements interns de la UC, la unitat de control hauria de quedar amb un aspecte semblant a aquest disseny:

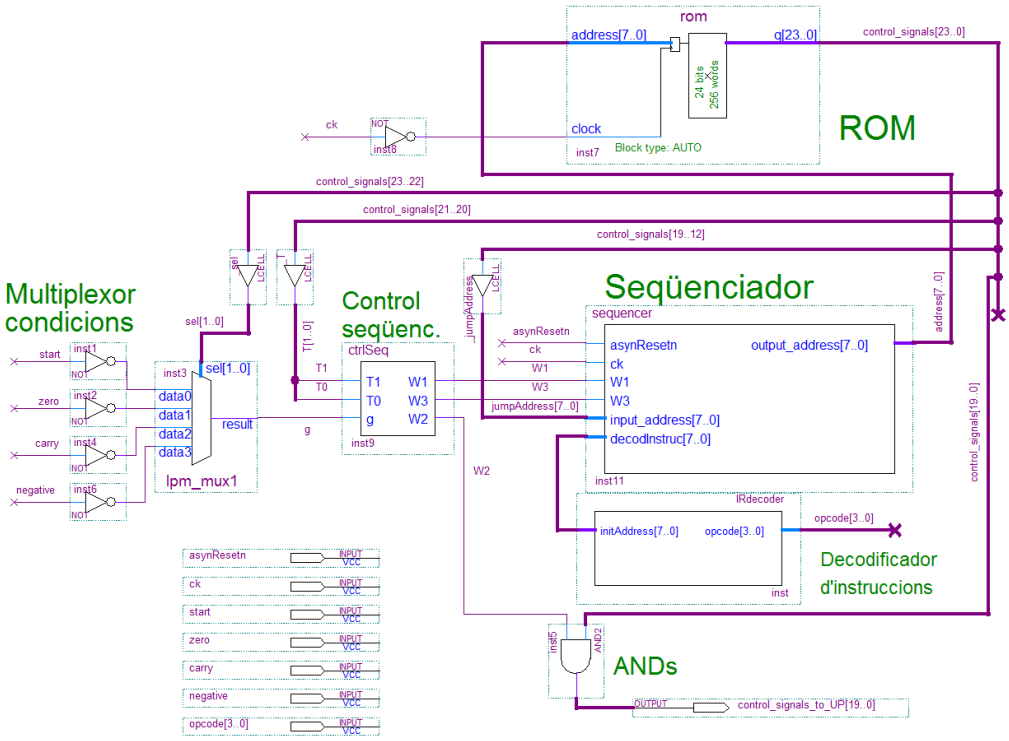


Fig. 8.12 Esquemàtic de la UC

Entrades

asynResetn: rèset de tots els registres (actiu a baixa).
ck: senyal de rellotge de tots els registres.
start: aquest bit permet que el processador comenci l'execució d'un programa.
zero: bit de condició que correspon a la sortida zero de la UP.
carry: bit que correspon a la sortida carry de la UP.
negative: bit que correspon a la sortida negative de la UP.
opcode[3..0] Indica *l'opcode* de la instrucció a executar.

Sortides

control_signals_to_UP[19..0] bits que provenen de la bateria de portes AND controlades pel senyal W2.

8.5. Simulació de la UC

Aquest punt s'ha inclós en aquest capítol dedicat a la UC per donar-li completesa. No obstant això, abans d'afrontar la simulació de la UC és necessari que llegiu els capítols 9 i 10 on s'explica de manera detallada el conjunt d'instruccions del processador i la seva implementació mitjançant microinstruccions a la ROM. Quan els hagueu llegits i entesos ja podeu dur a terme la simulació de la UC.

Una vegada dissenyada la UC, heu de simular-la per comprovar que el seu comportament és correcte. Us recomanem que prepareu una simulació amb retards d'acord amb les indicacions següents:

• Com a pas preliminar a la simulació, obriu la cel·la ctrlSeq i assigneu a les cel·les LCELL que generen els senyals de sortida W1, W2 i W3, els noms d'instància *_W1*, *_W2* i *_W3*, respectivament. Deseu la cel·la i recompileu la UC.

• La UC té com a única sortida el bus **control_signals_to_UP**. Aquest bus està format per l'agregació dels senyals de control generats per la UC. Els senyals de control ordenats des de la posició 19 del bus a la posició 0 són:

R1load, *R1out*, *R2load*, *R2out*, *IRload*, *IRout*, *MARload*, *MARout*, *MDRload*, *MDRout*, *MDRmem*, *MEMRW*, *PCload*, *PCout*, *PCinit*, *ACCload*, *ALUcode[2..0]*, i *end*.

Durant la simulació resulta més còmode treballar amb els noms dels senyals de control que amb els bits del bus control_signals_to_UP. A tal fi haureu de definir els noms dels senyals de control com a àlies dels bits del bus. El fitxer *alies.tcl* conté la definició dels àlies de tots els bits del bus control_signal_to_UP. Aquest fitxer us serà proporcionat i l'haureu de desar en el directori del projecte. Per establir els àlies heu d'executar l'ordre:

`Tools → Tcl Scripts...`

Finalment, seleccioneu el fitxer *alies.tcl* i premeu el botó **`Run`** (vegeu Fig. 8.13).

• Creeu un nou fitxer d'ones per especificar els vectors de simulació (**`File → New → Verification/Debugging Files → Vector Waveform File`**).

• Establiu 5 microsegons com a duració de la simulació (**`Edit → End Time`** 5us)

• Per facilitar el dibuix de les formes d'ona, establiu un engraellat de 50 ns (**`Edit → Grid Size`** 50 ns).

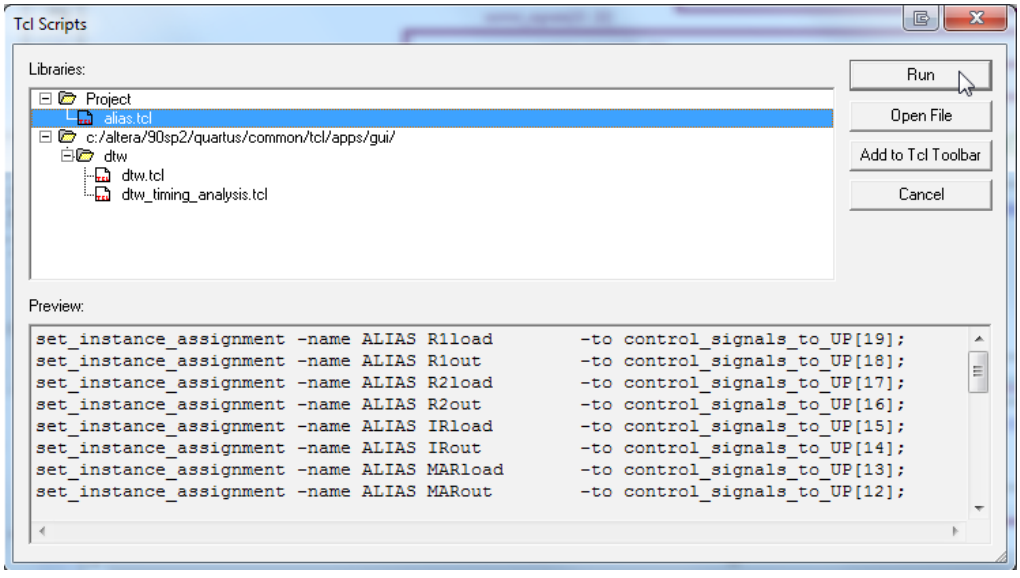


Fig. 8.13 Especificació dels àlies dels senyals de sortida

• A continuació hi ha una llista amb els senyals que heu d'afegir a l'editor de formes d'ona. Conté totes les entrades, totes les sortides i els nodes interns més importants. Al capítol 4 (secció 4.3.2) s'explicava com se seleccionen les entrades i com s'afegeixen a l'editor. A la finestra del **`Node Finder`**, per veure solament les entrades s'havia d'especificar al camp **`Filter`** l'opció **`Pins: Input`**. De la mateixa manera, si voleu veure solament les sortides heu d'especificar l'opció **`Pins: Output`**. Finalment, per accedir als nodes interns heu d'especificar l'opció **`Post-Compilation`**. La llista de senyals és la següent:

Entrades

asynResetn, carry, ck, negative, opcode (bus), start i zero.

Sortides

control_signals_to_UP (bus)

Nodes interns (on posa **instx***, la x representa un número d'un o dos dígits)*

ctrlSeq:instx|_W3	*sortida W3 de ctrlSeq*
ctrlSeq:instx|_W1	*sortida W1 de ctrlSeq*
ctrlSeq:instx|_W2	*sortida W2 de ctrlSeq*
IRdecoder:instx|initAddress	*bus de 8 bits ([7..0]); sortida de l'IRdecoder*
_jumpAddress	*bus de 8 bits ([7..0]); adreça de salt en les microinstruccions de tipus IF i GOTO*
_sel	*bus de 2 bits ([1..0]); bits de selecció de condició*
_T	*bus de 2 bits ([1..0]); bits de tipus de microinstrucció*
sequencer:instx|microPC:instx|mPC	*bus de 8 bits ([7..0]); contingut de microPC*
sequencer:instx|inc:instx|A_plus	*bus de 8 bits ([7..0]); sortida de l'incrementador*

• Substituïu els senyals de sortida pels seus àlies fent els passos següents:

- Obteniu els bits individuals del bus control_signals_to_UP. Per fer-ho, seleccioneu el bus i executeu l'ordre **`Grouping → Ungroup`**.
- Substituïu el nom de cada bit del bus pel seu àlies. Per editar un nom feu doble clic sobre ell. Recordeu que els àlies des de la posició 19 del bus a la posició 0 són:
R1load, *R1out*, *R2load*, *R2out*, *IRload*, *IRout*, *MARload*, *MARout*, *MDRload*, *MDRout*, *MDRmem*, *MEMRW*, *PCload*, *PCout*, *PCinit*, *ACCload*, *ALUcode_2*, *ALUcode_1*, *ALUcode_0* i *end*.
- Declareu els àlies com a senyals de tipus combinacional. Per fer-ho, seleccioneu tots els àlies (alhora), feu clic amb el botó dret del ratolí sobre la selecció i executeu l'ordre **`Properties`**. Al formulari que s'obre, al camp **`Type`** heu d'escollir **`COMB`**.
- Creeu un bus anomenat *ALUcode* amb els senyals *ALUcode_x*. Per fer-ho, seleccioneu els senyals ALUcode_2, ALUcode_1 i ALUcode_0, feu clic amb el botó dret del ratolí sobre la selecció i executeu l'ordre **`Grouping → Group...`**. Al formulari que es presenta introduïu *ALUcode* com a nom del grup.

• A continuació hi ha el guió de la simulació. Aquesta simulació serveix per comprovar el funcionament correcte de la UC durant l'execució de les instruccions LDR1, ADD, STR1 i BRE. En la instrucció BRE se simula tant el cas en què el senyal zero valgui 0 com el cas en què valgui 1. Fàcilment es podria ampliar la simulació per incorporar-hi altres instruccions. El rellotge **ck** inicialment val 0 i té un període de 100 ns amb un *duty cicle* del 50%.

t = 0 ns	Tots els senyals valen 0. El fet que asynResetn sigui 0 implica que s'està fent un *reset* de la unitat de control. El valor 0000 a opcode representa el codi de la instrucció LDR1.
t = 100 ns	asynResetn ← 1 (fi del *reset*)
t = 350 ns	start ← 1
t = 550 ns	start ← 0
t = 1,95 us	opcode ← codi de la instrucció ADD
t = 2,75 us	opcode ← codi de la instrucció STR1
t = 3,65 us	opcode ← codi de la instrucció BRE
t = 4,35 us	zero ← 1
t = 4,45 us	zero ← 0

• Per facilitar la llegibilitat dels resultats de la simulació, ordeneu els senyals d'acord amb la llista següent. Recordeu que, per ordenar els senyals, els heu de moure a la posició desitjada arrossegant-los amb el ratolí.

asynResetn	*entrada*
ck	*entrada*
start	*entrada*
opcode	*entrada*
carry	*entrada*
negative	*entrada*
zero	*entrada*
_sel	*bus de 2 bits ([1..0]); bits de selecció de condició*
_T	*bus de 2 bits ([1..0]); bits de tipus de microinstrucció*
R1load	*bit de sortida*
R1out	*bit de sortida*
R2load	*bit de sortida*
R2out	*bit de sortida*
IRload	*bit de sortida*
IRout	*bit de sortida*
MARload	*bit de sortida*
MARout	*bit de sortida*
MDRload	*bit de sortida*
MDRout	*bit de sortida*
MDRmem	*bit de sortida*
MEMRW	*bit de sortida*
PCload	*bit de sortida*
PCout	*bit de sortida*
PCinit	*bit de sortida*
ACCload	*bit de sortida*

ALUcode	*bus de sortida de 3 bits ([2..0]); operació de l'ALU*
end	*bit de sortida*
sequencer:instx|microPC:instx|mPC	*bus de 8 bits ([7..0]); contingut de microPC*
sequencer:instx|inc:instx|A_plus	*bus de 8 bits ([7..0]); sortida de l'incrementador*
_jumpAddress	*bus de 8 bits ([7..0]); adreça de salt en les microinstruccions de tipus IF i GOTO*
_IRdecoder:instx|initAddress	*bus de 8 bits ([7..0]); sortida de l'IRdecoder*
ctrlSeq:instx|_W3	*sortida W3 de ctrlSeq*
ctrlSeq:instx|_W1	*sortida W1 de ctrlSeq*
ctrlSeq:instx|_W2	*sortida W2 de ctrlSeq*

• Configureu el format de visualització dels busos. Per fer-ho, a l'editor d'ones posicioneu el ratolí sobre el bus la configuració del qual voleu establir, feu clic amb el botó dret i executeu **Properties → Radix → *Format desitjat***. Establiu el format de visualització dels busos de la manera següent:

- els busos interns sequencer:instx|microPC:instx|mP, sequencer:instx|inc:instx|A_plus, _jumpAddress i _IRdecoder:instx|initAddress en format decimal no signat.
- l'entrada opcode, els busos interns _sel i _T, i la sortida ALUcode en format binari.

• Deseu el fitxer d'ones amb el nom **UC.vwf**. Recordeu que heu d'especificar aquest fitxer en la configuració de la simulació (**Assignments → Settings → Simulator Settings → Simulation input**).

• Ara ja podeu llençar la simulació.

9. Conjunt d'instruccions

Tot seguit presentem el conjunt d'instruccions de *LittleProc*. A la taula següent s'indica l'*opcode* que correspon a cada instrucció i els paràmetres de cada instrucció.

Opcode	Instrucció	Paràmetres	Comentaris
0	LDR1 EXT	Posició de memòria	Carrega a R1 el valor de l'adreça de memòria indicada. *Ex.: a la posició 15 de la RAM hi ha el valor 128. Si es fa **LDR1 15**, o sense nemotècnic **00F** en HEX, s'estaria carregant a R1 el número 128.*
1	LDR2 EXT	Posició de memòria	Carrega a R2 el valor de l'adreça de memòria indicada. *Ex.: a la posició 16 de la RAM hi ha el valor 65. Si es fa **LDR2 16** (**110** en HEX), s'estaria carregant a R2 el número 65.*
2	ADD	Cap	Suma el contingut dels registres R1 i R2, i emmagatzema el resultat a R1. *Ex.: els valors emmagatzemats a R1 i R2 són 13 i 18, respectivament. Si es fa **ADD** (**200** en HEX), la suma de 13 i 18 (31) es carregaria a R1.*
3	STR1 EXT	Posició de memòria	Carrega a l'adreça de memòria indicada el valor de R1. *Ex.: a R1 hi ha el número 65. Si es fa **STR1 15** (**30F** en HEX), s'estaria carregant a la posició 15 de la RAM el valor 65.*
4	CMP	Cap	Compara si R1 i R2 són iguals. Si és així posa el *flag* de zero a '1'. *Ex.: R1 i R2 tenen tots dos el valor 19. Si es fa **CMP** (**400** en HEX), el flag **zero** adoptaria el valor 1. Si, a continuació, es carrega el valor 41 a R2 i es torna a executar **CMP**, el flag **zero** adoptaria el valor 0.*

Taula 9.1 ***Instruction set*** del processador (inici)

Opcode	Instrucció	Paràmetres	Comentaris
5	BRE EXT	Posició de memòria	Branch On Equal. Salta a l'adreça indicada si el *flag* de zero val '1'. Aquesta instrucció acostuma a anar acompanyada de la instrucció CMP. *Ex.: R1 i R2 tenen tots dos el valor 27. Si es fa **CMP** i després **BRE 33** (**521** en HEX), se saltaria a la posició 33 de la RAM.*
6	JMP EXT	Posició de memòria	Salt incondicional a l'adreça indicada. *Ex.: si es fa **JMP 50** (**632** en HEX), se saltaria a la posició 50 de la RAM.*
7	SUBS	Cap	Fa una resta entre R1 i R2, i emmagatzema el resultat a R1. *Ex.: els valors emmagatzemats a R1 i R2 són 17 i 11, respectivament. Si es fa **SUBS** (**700** en HEX), la resta de 17 i 11 (6) es carregaria a R1.*
8	END	Cap	Fa que s'activi el senyal *end* i el processador passi a estat d'espera. *Ex.: si es fa **END** (**800** en HEX), el senyal de sortida **end** tindria el valor 1 durant un cicle de rellotge i el processador passaria a un estat d'espera en el qual romandria fins que s'activés de nou el senyal d'entrada **start**.*
9	NOP	Cap	No fa res. El seu cicle d'execució consumeix un cicle de rellotge d'espera. *Ex.: si en un programa es vol deixar passar un cert temps sense fer res més, una manera de fer-ho és executar repetidament **NOP** (**900** en HEX).*
10	STR2	Posició de memòria	Carrega a l'adreça de memòria indicada el valor de R2. Ídem a STR1. *Ex.: a R2 hi ha el número 79. Si es fa **STR2 67** (**A43** en HEX), s'estaria carregant a la posició 67 de la RAM el valor 79.*

Taula 9.1 ***Instruction set*** del processador (continuació)

10. Disseny de la ROM de Microcodi

10.1. Estructura de la memòria de control

La memòria de control, que és el lloc on està emmagatzemat tot el codi de control de la UC (i d'on surten els senyals per la UP, i part de la UC), està ubicat a un fitxer extern que s'ha d'anomenar romv1q.mif (apartats 8.3.6 i 10.7).

Aquest fitxer segueix un estàndard hexadecimal creat per Intel i està format per dos grans apartats:

— Secció d'encapçalament on s'indica la grandària de la memòria en nombre de paraules i l'amplada de cada paraula, així com el format de les dades.

— El contingut de la taula seguint una estructura predefinida.

Tota l'estructura del fitxer és veurà amb molt més detall a l'apartat 10.7 després d'explicar l'organització interna d'aquesta ROM.

10.2. Seccions del microcodi de control

Tot seguit fem un breu esment de tota la funcionalitat que es pretén desenvolupar dins de la ROM de la UC. Aquesta està agrupada per apartats per fer un raonament més concret; a més dins de la Taula 10.1 se'n veurà amb una descripció molt més detallada.

10.3. Cicle d'inicialització

Té com a objectiu la inicialització del processador. Mentre que *start* no estigui actiu, el processador no queda inicialitzat. Tot seguit, una vegada inicialitzat es procedeix a posar un valor determinat (diferent per a cada grup de treball) en el PC (inicialització). Aquest procés consta de dues microinstruccions.

10.4. Cicle de *fetch* o de cerca de la instrucció

Aquest cicle és l'encarregat de prendre una instrucció de la memòria RAM i dur-la cap a l'IR (*Instruction Register*) per ser descodificada. Aquest procés consta, en principi, de cinc microinstruccions i està definit a la Taula 10.1.

A mode informatiu, la darrera microinstrucció (de tipus BRA) té la funció de fer la càrrega en el seqüenciador de l'adreça d'inici del cicle d'execució de la instrucció actual. Aquesta adreça és generada per l'IRdecoder a partir de *l'opcode* emmagatzemat a l'IR.

10.5. Cicles d'execució de les diverses instruccions

Conjunt de microinstruccions que implementen els cicles d'execució de totes les instruccions del processador. Les microinstruccions són executades segons indica el microPC. Tot seguit mostrem la taula de correspondència de la nostra Unitat de Control emmagatzemada a la ROM.

						//////////		Espai per a l'adreça de salt					//////////												
Adr.	`	sel1	sel0	T1	T0	R1load	R1out	R2load	R2out	IRload	IRout	MARload	MARout	MDRload	MDRout	MDRmem	MEMRW	PCload	PCout	PCinit	ACCload	ALU2	ALU1	ALU0	end
	INICIALITZACIÓ PC																								
0	IF (not start) GOTO 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	EXE(INIT PC)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
	FETCH																								
2	EXE(MAR<- PC)	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
3	EXE(read RAM)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	EXE(MDR<- RAM[MAR], PC++)	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	0	0	1	0	0	0
5	EXE (IR<- MDR)	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
6	BRA(JMP $Cicle _Exec.)	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	NOP																								
7	GOTO 2	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	LDR1																								
8	EXE(MAR<- Mask(IR))	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0
9	EXE(read RAM)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	EXE(MDR<- RAM[MAR])	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0
11	EXE(R1<-MDR)	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
12	GOTO 2	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	LDR2																								
13	FER LDR2																								
14	FER LDR2																								
15	FER LDR2																								
16	FER LDR2																								
17	FER LDR2																								
	ADD																								
18	EXE(ACC<-R1)	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
19	EXE(R1<- ACC+R2)	0	0	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
20	GOTO 2	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	STR1																								
21	EXE(MAR<- Mask(IR))	0	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0
22	EXE(MDR<-R1)	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
23	EXE(RAM write)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
24	GOTO2	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	CMP																								
25	EXE(ACC<-R1)	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
26	EXE(ACC-R2)	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
27	GOTO 2	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	BRE																								
28	IF (not zero) GOTO 2	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
29	EXE(PC<-Mask(IR))	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	1	0
30	GOTO 2	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Taula 10.1 Taula de correspondència de la ROM (inici)

						//////////		Espai per a l'adreça de salt					//////////												
Adr.	`	sel1	sel0	T1	T0	R1load	R1out	R2load	R2out	IRload	IRout	MARload	MARout	MDRload	MDRout	MDRmem	MEMRW	PCload	PCout	PCinit	ACCload	ALU2	ALU1	ALU0	end
	JMP																								
31	FER JUMP																								
32	FER JUMP																								
	SUBS																								
33	FER SUBST																								
34	FER SUBST																								
35	FER SUBST																								
	STR2																								
36	FER STR2																								
37	FER STR2																								
38	FER STR2																								
39	FER STR2																								
	END																								
40	FER END																								
41	FER END																								

Taula 10.1 Taula de correspondència de la ROM (continuació)

Microcodi:

Línies 0 – 1 : Inicialització
Línies 2 – 6 : Cicle de cerca de la instrucció (fetch)
Línia 7 : Cicle d'execució de la instrucció NOP
Línies 8 – 12 : Cicle d'execució de la instrucció LDR1
Línies 13 – 17 : Cicle d'execució de la instrucció LDR2
Línies 18 – 20 : Cicle d'execució de la instrucció ADD
Línies 21 – 24 : Cicle d'execució de la instrucció STR1
Línies 25 – 27 : Cicle d'execució de la instrucció CMP
Línies 28 – 30 : Cicle d'execució de la instrucció BRE
Línies 31 – 32 : Cicle d'execució de la instrucció JMP
Línies 33 – 35 : Cicle d'execució de la instrucció SUBS
Línies 36 – 39: Cicle d'execució de la instrucció STR2
Línies 40 – 41: Cicle d'execució de la instrucció END

10.6. Taula de descodificació

D'aquesta manera, havent presentat totes les instruccions i microinstruccions, podem determinar definitivament la taula de descodificació implementada mitjançant el mòdul IRDecoder. Recomanem repassar la taula original i afegir/modificar els canvis necessaris per tal de considerar totes les instruccions.

Opcode	Adreça
0000	00001000
0001	00001101
0010	00010010
0011	00010101
0100	00011001
0101	00011100
0110	00011111
0111	00100001
1000	00101000
1001	00000111
1010	00100100

Taula 10.2 Taula de descodificació de les instruccions

10.7. Contingut de la ROM

En aquesta secció presentarem part del microcodi emmagatzemat dins de la ROM. El contingut de la ROM és especificat mitjançant el fitxer romv1q.mif (Fig. 10.1). Aquest fitxer té dues parts diferenciades:

- una capçalera amb informació descriptiva de la ROM
- el contingut de les paraules de la ROM

La capçalera segueix la sintaxi següent:

DEPTH = nombre de paraules de la ROM;
WIDTH = nombre de bits de cada paraula de la ROM;
ADDRESS_RADIX = format en què s'escriuen les adreces a la segona part del fitxer;
DATA_RADIX = format en què s'escriuen les dades a la segona part del fitxer;

Els formats s'indiquen mitjançant les cadenes *BIN* (binari), *DEC* (decimal), *HEX* (hexadecimal), *OCT* (octal) i *UNS* (decimal no signat). En el nostre cas, expressarem les adreces en format decimal i les dades en format binari.

En un fitxer .mif, els dobles guionets (--) indiquen que el que hi ha fins al final de la línia és un comentari. També és possible emmarcar els comentaris entre signes %.

En la segona part del fitxer (sentència *CONTENT BEGIN END;*) s'indica el contingut de cada paraula de la ROM (es a dir, cada microinstrucció) seguint la sintaxi següent:

adreça en decimal : contingut de la paraula en binari;

Quan un conjunt de paraules consecutives tenen el mateix contingut es pot indicar mitjançant la sintaxi següent:

[adreça inicial en decimal .. adreça final en decimal] : contingut en binari;

El contingut de les posicions de memòria l'escrivim en binari per tal de facilitar la identificació del valor de cada senyal.

D'altra banda cal indicar que el valor binari de cada posició no es pot fragmentar en bits (separar els bits amb espais) perquè es pugui adjudicar visualment a una columna del senyal i que així sigui més fàcil omplir-la. Els fitxers .mif han de tenir el valor en binari tot en una peça i no els podem fragmentar. Per tant:

00000000 és correcte.
0 0 0 0 0 0 0 0 no és correcte.

Així doncs, a la Fig. 10.1 presentem part d'aquesta ROM. La part que falta s'ha de desenvolupar com a exercici.

Observem que la grandària de la ROM és de 256 paraules, i que cada paraula està formada per 24 bits (tants com senyals hem de controlar).

```
DEPTH = 256;                % Memory depth and width are required        %
WIDTH = 24;                 % Enter a decimal number                     %
ADDRESS_RADIX = DEC;        % Address and value radixes are required     %
DATA_RADIX = BIN;           % Enter BIN, DEC, HEX, OCT, or UNS; unless %
                            % other w is specified, radixes=HEX          %

-- Specify values for addresses, which can be single address or range
CONTENT
BEGIN
--  sel(2) T(2) / R1load R1out R2load R2out IRload IRout MARload MARout /
MDRload MDRout MDRmem MEMRW PCload PCout PCinit ACCload ALUcode(3) end
-- inicialització
0     : 000000000000000000000000;
1     : 001000000000000010100000;
-- cicle de cerca
2     : 001000000010000001000000;
3     : 001000000000000000000000;
4     : 001000000000101011001000;
5     : 001000001000010000000000;
6     : 001100000000000000000000;
```

TOTA AQUESTA PART S'HA D'OMPLIR AMB ELS VALORS DE LA TAULA DE CORRESPONDÈNCIA

```
[42..255]: 000000000000000000000000;
END;
```

Fig. 10.1 Fitxer de la ROM

11. Disseny de la RAM

11.1. Estructura de la Memòria RAM

La memòria RAM d'instruccions/dades està bàsicament formada per un bus d'adreces, un bus de dades, un bit Read/Write i un senyal de Clock, a més d'un bus de dades de sortida.

Aquesta memòria és l'encarregada d'emmagatzemar cada una de les instruccions a ser executades pel processador.

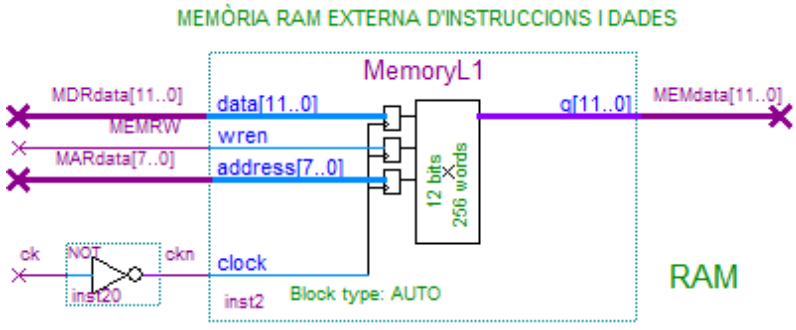


Fig. 11.1 Memòria RAM d'instruccions/dades

Entrades

- **data[11..0]** connectat al Bus MDRdata[11..0] que surt de la UP (bus de dades).
- **wren** bit que indica si s'està llegint o escrivint en RAM, connectat a MEMRW. Aquest senyal és anàleg a control_signals_to_up[8] i s'obté fent passar aquest últim per un *buffer* (LCELL).
- **address[7..0]** connectat al bus MARdata[7..0] que surt de la UP (bus d'adreces).
- **clock**. Senyal de rellotge connectat al CK genèric, però invertit.

Sortides

- **q[11..0]** connectat a l'entrada MEM[11..0] de la UP mitjançant el bus MEMdata[11..0].

11.2. Creació de la Memòria RAM

Per tal de dissenyar la memòria RAM, de la mateixa manera que hem dissenyat abans els multiplexors i la ROM, fem servir el mateix WIZARD (*Tools/MegaWizard Plug- In Manager...)* i tot seguit fem el procés vist a l'apartat 7.5.1.3 del MDR, però tenint en

compte que ara el component es troba a la secció *Memory Compiler* i model *RAM:1-PORT*.

Respecte als atributs a l'hora de crear-la, cal considerar que l'amplada del bus de dades ha de ser de 12 bits, i que el nombre total de paraules de 12 bits ha de ser de 256 (espai de memòria).

D'altra banda el tipus de Block de memòria el fixem en mode "auto" i el tipus de rellotge ha de ser "*single clock*"; no hem de marcar res més (desactiveu l'opció *'q' output port*), però ens hem d'assegurar que el nom del fitxer és ramv1.mif.

11.3. Estructura de les dades

La informació es desa dins d'aquesta memòria ordenada per posicions, tenint en compte que el mètode per escriure el programa es basa en el model que mostrem tot seguit.

En el model podem observar una capçalera que indica:

- La llargada en posicions de la RAM (256).
- El nombre de bits de cada posició (12).
- El format amb el qual s'escriuen les adreces (DEC = decimal).
- El format amb el qual s'escriuen les dades (HEX = hexadecimal).

Els dobles guionets indiquen que el que ve a continuació és un comentari igual que els %.

```
DEPTH = 256;                % Memory depth and width are required        %
WIDTH = 12;                 % Enter a decimal number                     %
ADDRESS_RADIX = DEC;        % Address and value radixes are required     %
DATA_RADIX = HEX;           % Enter BIN, DEC, HEX, OCT, or UNS; unless   %
                            % otherwise specified, radixes = HEX         %

-- Contingut de la memòria RAM
CONTENT
BEGIN

0       :   0FF; % Comentari %
[1..255]:   000;
END;
```

Fig. 11.2 Fitxer de la RAM: especificació de la capçalera.

El contingut de cada posició de la RAM s'expressa mitjançant la sintaxi següent:

Posició RAM en DEC: Contingut de la posició en HEX % Comentari, si cal %

Respecte a la posició de la RAM, veiem que ha d'anar entre el valor 0 i el màxim, que és 255; es presenta en decimal.

Respecte al contingut de la posició escrita en hexadecimal, cal destacar que està format per tres xifres en què:

- La primera (més a l'esquerra) indica l'opcode de la instrucció que s'ha d'executar.
- Les dues xifres restants són l'operand de la instrucció; per exemple:
  - si es tracta d'un salt incondicional, amb aquestes dues xifres indiquem l'adreça de salt.
  - si es tracta d'una càrrega al R1 d'una posició de memòria, LDR1, aquest camp indica l'adreça d'aquesta posició.

D'aquesta manera, i a mode d'exemple, si volem fer un programa que agafi la dada de la posició de memòria 15 i la posi a R1, que tot seguit agafi la dada de la posició 16 i la posi a R2, que a continuació faci la suma de R1 i R2, desi el resultat a la posició de memòria 17 i finalment activi el senyal *end*, hauríem d'especificar aquest programa dins del fitxer ramv1.mif de la manera següent:

```
DEPTH = 256;                % Memory depth and width are required                %
WIDTH = 12;                 % Enter a decimal number                             %
ADDRESS_RADIX = DEC;        % Address and value radixes are required             %
DATA_RADIX = HEX;           % Enter BIN, DEC, HEX, OCT, or UNS; default = HEX    %

CONTENT
BEGIN
0       :   000;
1       :   00F;    % 000000001111   LDR1 15          %
2       :   110;    % 000100010000   LDR2 16          %
3       :   200;    % 001000000000   ADD              %
4       :   311;    % 001100010001   STR1 17          %
5       :   800;    % 100000000000   END              %
6       :   000;
[7..14]  :  000;
15      :   001;    % A partir d'aquí van les dades   %
16      :   002;
17      :   000;
18      :   000;
[19..255]:  000;
END;
```

Fig. 11.3 Fitxer de la RAM: especificació d'un programa d'exemple

11.4. Funcionament de la RAM

En els cronogrames següents queda resumit el funcionament de la RAM segons els seus senyals de control. En el primer cronograma es pot apreciar un procés de lectura, el qual es realitza regit pel flanc de pujada del rellotge i amb el senyal *wren* desactivat (valor 0). Nota: el senyal *wren* no apareix en el cronograma.

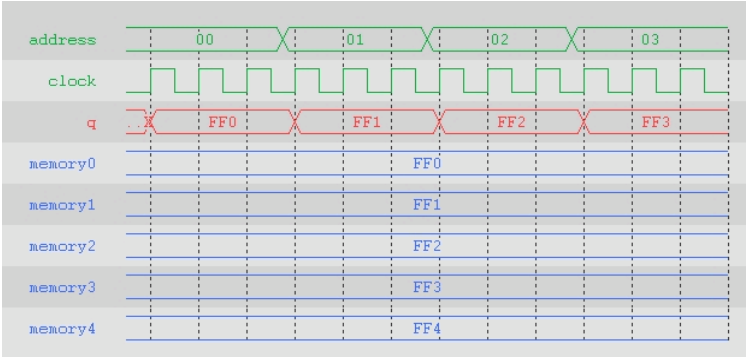


Fig. 11.4 Cronograma amb cicles de lectura de la RAM

En el cronograma següent es pot observar una operació d'escriptura. El cicle d'escriptura comença en el flanc de pujada del rellotge en el qual el senyal *wren* és actiu (el seu valor és 1) i acaba en el següent flanc de pujada del rellotge. Dins d'aquest cicle, l'escriptura pròpiament dita es fa en el flanc de baixada del rellotge. La dada quan s'escriu a la memòria també resulta visible a través del terminal de sortida q<11:0>.

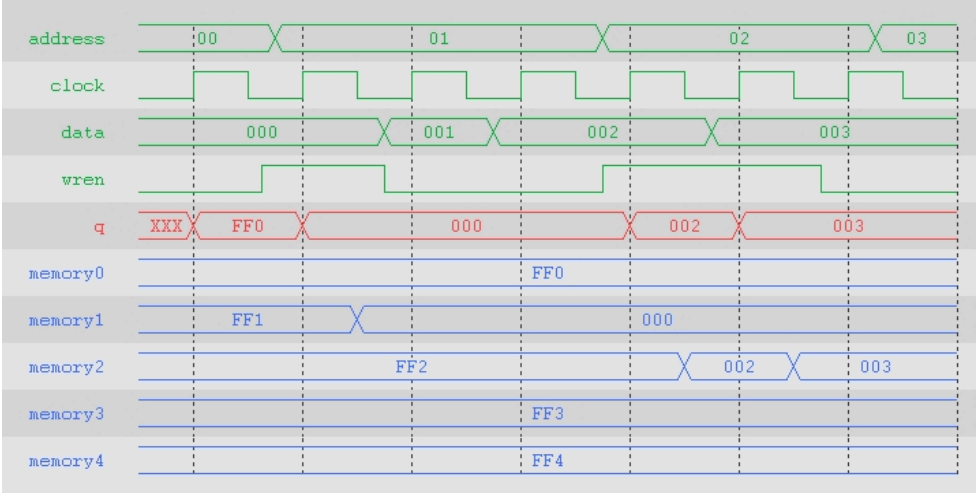


Fig. 11.5 Cronograma amb un cicle d'escriptura a la RAM

12. Disseny intern del processador

Finalment presentarem l'estructura interna final del processador, del qual recordarem els elements pels quals està format:

- Una memòria RAM de dades/programa que conté el programa a executar i una zona de memòria reservada per a dades (variables).

- Una unitat de control que és l'encarregada de generar la seqüència d'ordres necessàries per executar cada instrucció emmagatzemada a la RAM interna.

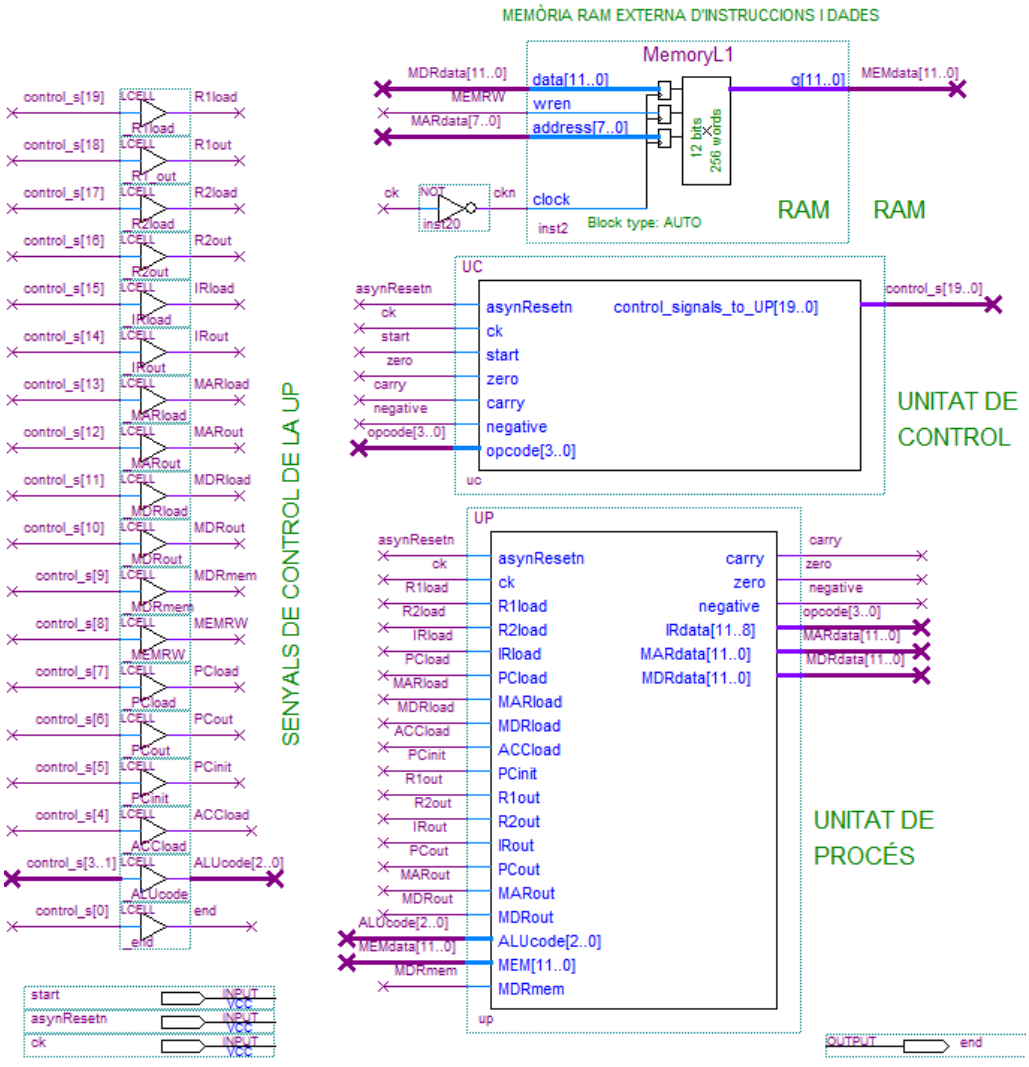


Fig. 12.1 Esquema global del processador

- La unitat de procés, que és l'encarregada de processar totes les dades, seguint les ordres de la unitat de procés.

- Un seguit de *buffers* LCELL, que permetin la monitorització correcta dels senyals de control de la UP per tal de fer la simulació final.

Expansor del bus de control

Finalment, per tal de poder fer visible en el simulador cadascun dels senyals de control sobre la UP, i per tal de fer més entenedor el disseny del processador, hem decidit segmentar tot el bus de sortida (control_s[19..0]) en un seguit de bits independents que es mostren a la Taula 12.1.

Cada bit es un senyal de control i es genera amb una cel·la LCELL. El nom d'instància de la cel·la LCELL és el mateix nom del senyal, però prefixat amb un guió baix. Per exemple, la cel·la LCELL que genera el senyal *R1load* té com a nom d'instància *_R1load*. Quan simuleu el processador, veureu que per fer referència al senyal *R1load* fareu servir el nom d'instància de la cel·la LCELL que el genera, és a dir, *_R1load*.

- control_s[19] a R1load
- control_s[18] a R1out
- control_s[17] a R2load
- control_s[16] a R2out
- control_s[15] a IRload
- control_s[14] a IRout
- control_s[13] a MARload
- control_s[12] a MARout
- control_s[11] a MDRload
- control_s[10] a MDRout
- control_s[9] a MDRmem
- control_s[8] a MEMRW
- control_s[7] a PCload
- control_s[6] a PCout
- control_s[5] a PCinit
- control_s[4] a ACCload
- control_s[3..1] a ALUcode [2..0]
- control_s[0] a end

Taula 12.1 Mapeig de senyals de control

Entrades

start : senyal que permet posar en funcionament tota la CPU.

asynResetn : senyal que fa el rèset asíncron de tot el sistema. És necessari fer sempre un rèset al principi del funcionament del processador.

ck : senyal de sincronisme de tota la CPU.

Sortides

end : senyal que es posa a '1' al final de tota l'execució.

12.1. Simulació del processador

Com a pas preliminar a la simulació, assegureu-vos que les cel·les UC i UP tenen com a nom d'instància *uc* i *up*, respectivament. De la mateixa manera, totes les cel·les LCELL han de tenir com a nom d'instància el mateix nom del senyal que generen prefixat amb un guió baix. Si feu canvis al circuit, deseu el circuit i recompileu-lo.

Una vegada dissenyat tot el procesador, heu de simular-lo per comprovar que el seu comportament és correcte. En aquest cas, no us proporcionarem indicacions sobre com s'ha de preparar la simulació. Amb l'experiència que heu adquirit fent les simulacions de la UP i de la UC hauríeu de saber com dur a terme la simulació. Tan sols us indiquem els senyals que hauríeu de visualitzar a la simulació. Recordeu que en els noms d'alguns nodes interns, on posa **instx**, la **x** representa un número d'un o dos dígits.

asynResetn	*entrada*
ck	*entrada*
start	*entrada*
_PCinit	*node intern; senyal d'inicialització del PC*
_PCload	*node intern; senyal de càrrega del PC*
PCdataout	*node intern; contingut del PC. Aquest node no el trobareu amb aquest nom: l'heu de construir com ja vau fer a la simulació de la UP*
_PCout	*node intern; senyal de pas del PC al Bus 1*
_MEMRW	*node intern; senyal de lectura/escrit. de la RAM*
_MARload	*node intern; senyal de càrrega del MAR*
UP:up|reg:MAR|dataout	*node intern; contingut del MAR*
_MARout	*node intern; senyal de pas del MAR al Bus 1*
_MDRload	*node intern; senyal de càrrega del MDR*
UP:up|reg:MDR|dataout	*node intern; contingut del MDR*
_MDRout	*node intern; senyal de pas del MDR al Bus 1*
_IRload	*node intern; senyal de càrrega de l'IR*
UP:up|reg:IR|dataout	*node intern; contingut de l'IR*
_IRout	*node intern; senyal de pas de l'IR al Bus 1*
_R1load	*node intern; senyal de càrrega de R1*
UP:up|reg:R1|dataout	*node intern; contingut de R1*
_R1out	*node intern; senyal de pas de R1 al Bus 1*
_R2load	*node intern; senyal de càrrega de R2*
UP:up|reg:R2|dataout	*node intern; contingut de R2*
_R2out	*node intern; senyal de pas de R2 al Bus 1*
_ACCload	*node intern; senyal de càrrega d'ACC*
UP:up|reg:ACC|dataout	*node intern; contingut d'ACC*

UC:uc|sequencer:instx|microPC:instx|mPC	*node intern; contingut del microPC*
UC:uc|sequencer:instx|inc:instx|A_plus	*node intern; sortida de l'incrementador*
UC:uc|_jumpAddress	*node intern; adreça de salt del seqüenciador*
UC:uc|IRdecoder:instx|initAddress	*node intern; sortida de l'IRdecoder*
UC:uc|_sel	*node intern; bits de selecció de condició*
UC:uc|_T	*node intern; bits de tipus de microinstrucció*
UC:uc|ctrlSeq:instx|_W3	*node intern; bit de control del seqüenciador*
UC:uc|ctrlSeq:instx|_W1	*node intern; bit de control del seqüenciador*
UC:uc|ctrlSeq:instx|_W2	*node intern; senyal que controla el pas a l'exterior dels senyals produïts per la UC*
_ALUcode	*node intern: senyals de control de l'ALU*
UP:up|ALU:ALU|C	*node intern; sortida C de l'ALU*
UP:up|ALU:ALU|zero	*node intern; sortida zero de l'ALU*
UP:up|ALU:ALU|carry	*node intern; sortida carry de l'ALU*
UP:up|ALU:ALU|negative	*node intern; sortida negative de l'ALU*
end	*sortida*

13. Disseny del microprocessador amb un bus

Com ja s'ha vist anteriorment la unitat de procés té la funció d'emmagatzemar el conjunt de registres que s'encarreguen del funcionament correcte de l'execució d'instruccions, així com de realitzar les operacions necessàries. En aquest cas, en comptes de fer servir 2 busos, es farà servir un bus tant per a l'entrada de les dades com per a les sortides. El muntatge és similar a l'explicat amb anterioritat. El que canvia són les microinstruccions, ja que ara les operacions necessiten més cicles per ser realitzades.

La nostra unitat de procés està formada per:

— Un conjunt de registres d'emmagatzemament.
— Un multiplexor.
— 1 bus.
— Una ALU.

En el gràfic següent en podem observar l'estructura interna.

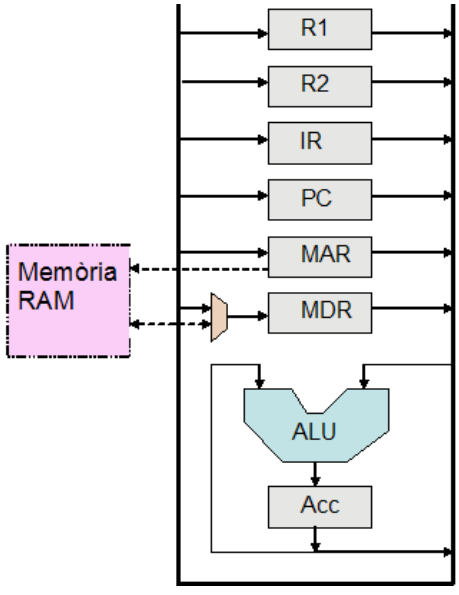


Fig. 13.1 Diagrama de la unitat de procés

13.1. Arquitectura genèrica de la UP

Atesa la relació de registres, busos i elements interns, la UP hauria de quedar amb un aspecte semblant al disseny de la Fig. 13.2.

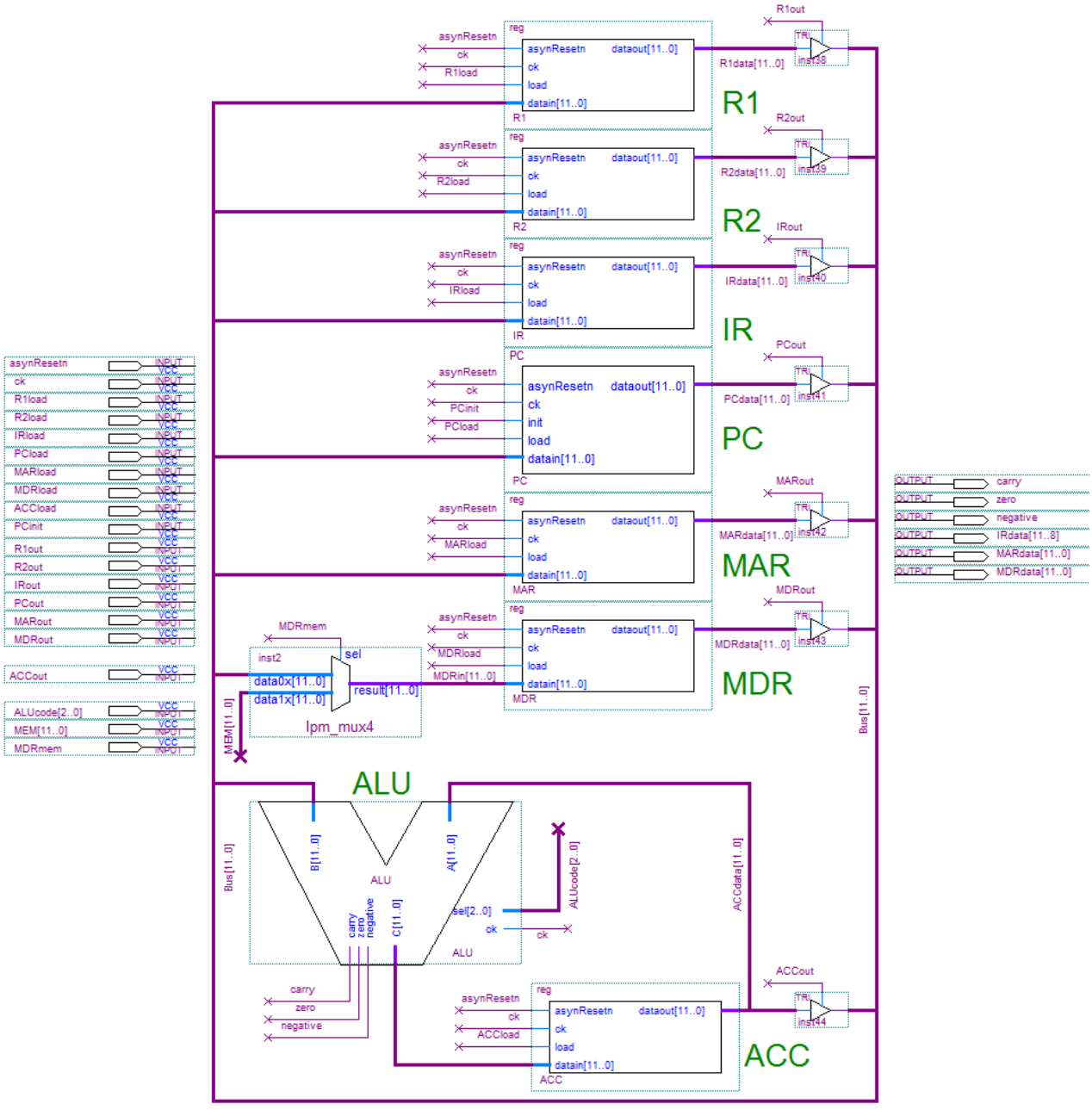


Fig. 13.2 Unitat de procés d'un sol bus

13.2. Unitat de control. Microcodi

En aquest cas, el cicle d'inicialització, el cicle de *fetch* o de cerca de la instrucció i el cicle d'execució de les diverses instruccions tenen petites variacions com es mostra a la taula següent:

						//////////		Espai per a l'adreça de salt					//////////													
Adr.	`	sel1	sel0	T1	T0	R1load	R1out	R2load	R2out	IRload	IRout	MARload	MARout	MDRload	MDRout	MDRmem	MEMRW	PCload	PCout	PCinit	ACCload	ACCout	ALU2	ALU1	ALU0	end
	INICIALITZACIÓ PC																									
0	IF (not start) GOTO 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	EXE(INIT PC)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0
	FETCH																									
2	EXE(ACC<- PC)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
3	EXE(MAR<-ACC)	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
4	EXE(read RAM)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	EXE(MDR<- RAM), ACC<- ++PC	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	1	0	1	0	0	0
6	EXE(PC<-ACC)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
7	EXE(ACC<- MDR)	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0
8	EXE (IR<- ACC)	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
9	BRA(.JMP $Cicle_Exec.)	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	NOP																									
10	GOTO 2	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	LDR1																									
11	EXE(ACC<- Mask(IR))	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0
12	EXE(MAR<- ACC)	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
13	EXE(read RAM)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	EXE(MDR<- RAM)	0	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
15	EXE(R1<-MDR)	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
16	GOTO 2	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	LDR2																									
17	FER LDR2																									
18	FER LDR2																									
19	FER LDR2																									
20	FER LDR2																									
21	FER LDR2																									
22	FER LDR2																									
	ADD																									
23	EXE(ACC<-R1)	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
24	EXE(ACC<- ACC+R2)	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
25	EXE(R1<-ACC)	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
26	GOTO 2	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	STR1																									
27	EXE(ACC<-Mask(IR))	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0
28	EXE(MAR<-ACC)	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
29	EXE(MDR<-R1)	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
30	EXE(RAM write)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
31	GOTO2	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Taula 13.1 Taula de la ROM (inici)

Adr.	`	sel1	sel0	T1	T0	R1load	R1out	R2load	R2out	IRload	IRout	MARload	MARout	MDRload	MDRout	MDRmem	MEMRW	PCload	PCout	PCinit	ACCload	ACCout	ALU2	ALU1	ALU0	end
	CMP																									
32	EXE(ACC<-R1)	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
33	EXE(ACC-R2)	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0
34	GOTO 2	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	BRE																									
35	IF (not zero) GOTO 2	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
36	EXE(ACC<-Mask(IR))	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0
37	EXE(PC<-ACC)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
38	GOTO 2	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	JMP																									
39	FER JUMP																									
40	FER JUMP																									
41	FER JUMP																									
	SUBS																									
42	FER SUBST																									
43	FER SUBST																									
44	FER SUBST																									
45	FER SUBST																									
	STR2																									
46	FER STR2																									
47	FER STR2																									
48	FER STR2																									
49	FER STR2																									
50	FER STR2																									
	END																									
51	FER END																									
52	FER END																									

Taula 13.1 Taula de la ROM (continuació)

Taula d'instruccions:

Línies 0 – 1 : INICIALITZACIÓ
Línies 2 – 9 : Cicle de cerca de la instrucció (FETCH)
Línia 10 : NOP
Línies 11 – 16 : LDR1
Línies 17 – 22 : LDR2
Línies 23 – 26 : ADD
Línies 27 – 31 : STR1
Línies 32 – 34 : CMP
Línies 35 – 38 : BRE
Línies 39 – 41 : JMP
Línies 42 – 45 : SUBS
Línies 46 – 50: STR2
Línies 51 – 52: END

14. Implementació a la placa Altera UP3 Education Kit

Per implementar el nostre disseny dins una FPGA en general, hem de tenir en compte diverses etapes. Primer s'ha de pensar que la FPGA és un xip que té una sèrie de pins d'entrada i altres de sortida. Per tant, hem de tenir clar com entrar la nostra informació dins del xip. Normalment la FPGA està muntada dins una placa. Aquesta placa té una sèrie d'interfícies per actuar-hi. Hi ha, d'una banda, interfícies d'entrada, ja sigui teclat, ratolí, etc. També hi ha senyals de sortida que es poden visualitzar a través d'una pantalla led, o a través de busos de sortida de tipus USB, ps2, etc.

Per tant, és molt important tenir en compte que s'ha de fer una assignació de les entrades/sortides del nostre disseny (LittleProc) amb els pins de la FPGA que estan connectats a la placa de prototipatge amb les interfícies que ens interessen. Aquesta assignació s'acostuma a fer a partir d'un fitxer que assigna cadascuna de les entrades/sortides del sistema amb pins físics de la FPGA.

Dins una placa hi ha un oscil·lador que és el que dóna el rellotge a la placa; aquest normalment va a Mhz. Hem de tenir clar que aquest oscil·lador ens dóna el rellotge del nostre sistema. S'ha de saber quin és el pin per on entra aquest senyal. A més, si volguéssim veure com oscil·la, per exemple, un led a aquesta freqüència, ens seria impossible, ja que l'ull humà no pot discernir si un led s'apaga o s'encén si la freqüència és aproximadament major de 24Hz. Per tant, per al nostre disseny s'han d'adequar els senyals per poder-los visualitzar.

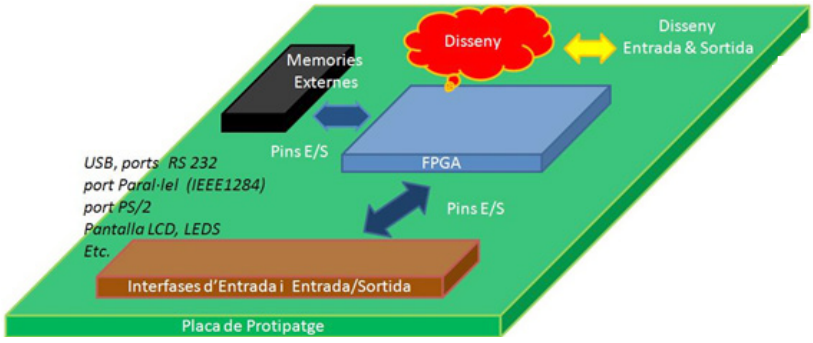


Fig. 14.1 Esquema bàsic d'una placa amb una FPGA

Als apartats següents s'explica un cas concret de com implementar el nostre disseny a una placa de prototipatge. Fer-ho amb un altre seguiria una filosofia similar de:

- Adequar els senyals del nostre disseny amb els pins de la FPGA.
- Tenir els *drivers* de com comunicar aquests senyals amb els busos d'entrada/sortida. Això ho aconseguim fent servir els *datasheets* (documentació relativa de la placa).
- Adequar les freqüències dels senyals, si calgués, per tal de poder veure els resultats.

14.1. Placa de prototipatge

La placa (Fig. 14.2) té les característiques i els components següents:

— *Placa per a dissenys amb FPGA*
— *Dispositiu EP1C12Q240 (o EP1C6Q240)*
— *Dispositiu de configuració EPCS4 (o EPCS1)*
— *Suporta dissenys basats en IP (propietats intel·lectuals), tant amb microprocessador o sense*
— *Interconnexions seguint estàndards industrials com:*
  · *USB 2.0-compliant (total i baixa velocitat)*
  · *Dos ports RS 232*
  · *Un port paral·lel (IEEE1284)*
  · *Un port PS/2*
— *Memòria externa del sistema*
  · *1 Mbyte de SRAM*
  · *2 Mbytes de FLASH*
  · *2 Kbytes d'I2C PROM (expansible)*
— *Múltiples rellotges per al disseny del sistema*
— *Configuració del JTAG*
— *Pins d'expansió per a una major flexibilitat i capacitat*
— *Targeta d'expansió de 5V Santa Cruz que proporciona 72 I/O pins per a un desenvolupament, connexió addicional per connectar diverses plaques (proporcionen diferents funcionalitats)*
— *Característiques addicionals que es proporcionen a l'usuari*
  · *Bloc de commutadors; 4 bits que poden ser definits per l'usuari*
  · *Quatre botons polsadors que poden ser definits per l'usuari i un commutador que fa de reset global*
  · *Quatre leds*
  · *Un mòdul de display LCD de 16x2 caràcters*
  · *Un rellotge I2C a temps real*

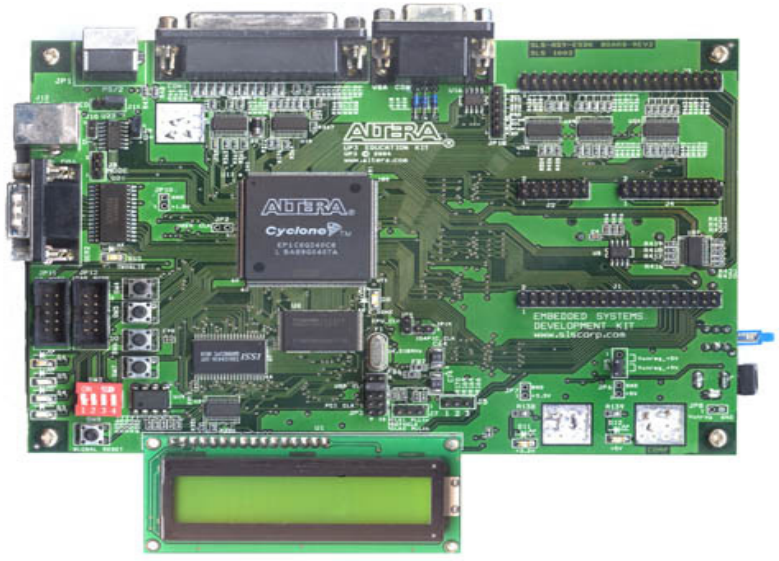


Fig. 14.2 Placa de prototipatge UP3 Education Kit

14.2. Modificació de LittleProc

La implementació en placa de LittleProc està basada en tres canvis bàsics, seguint les premisses explicades a l'apartat final de la pàgina 143:

1. Afegir un arxiu amb l'assignació de terminals d'entrada i sortida del nostre circuit amb els pins de la FPGA.
2. Afegir un mòdul controlador de la pantalla LCD.
3. Posar un rellotge amb freqüència baixa; en el nostre cas es fan els polsos de manera manual amb un polsador manual.

La Fig. 14.3 mostra una visió del processador sense aquestes modificacions tal com queda un cop finalitzat el capítol 12 i veient que ha funcionat. A la Fig. 14.4 es pot observar com quedaria el processador una vegada modificat.

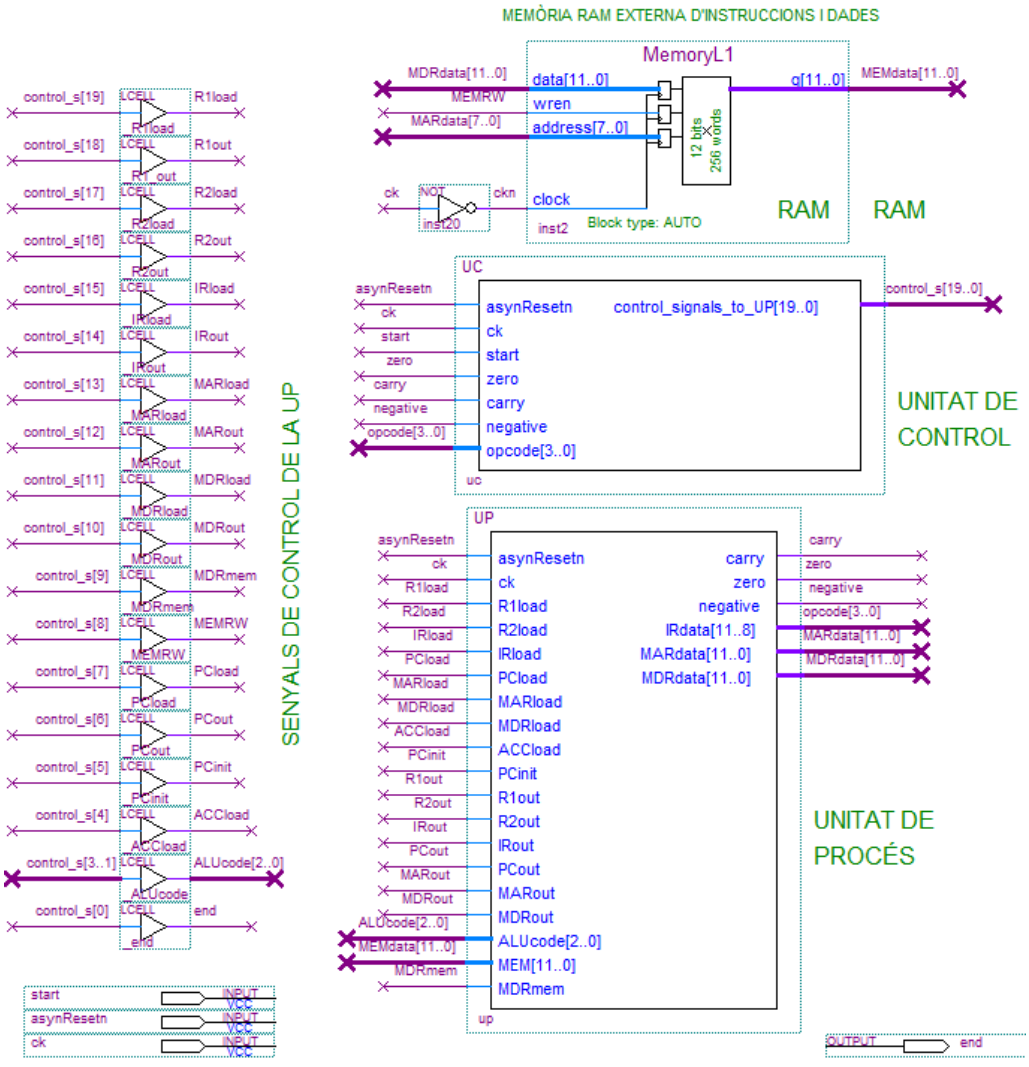


Fig. 14.3 Visió de l'esquema de LittleProc

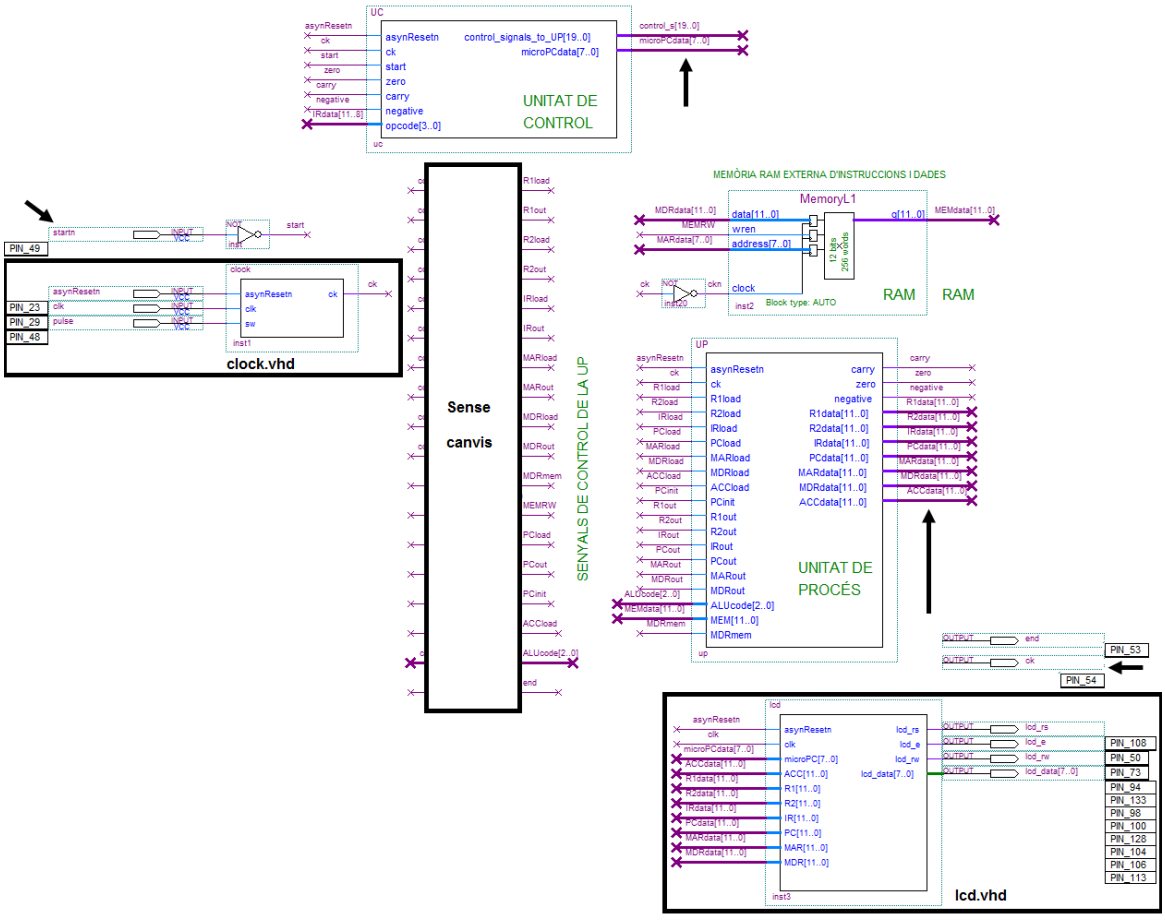


Fig. 14.4 Modificacions de ***LittleProc*** per implementar-lo a placa

En els apartats següents aprofundim en com es modifica el nostre circuit per poder-lo ficar a la placa. Per fer la implementació correcta del microprocesador es necessiten una sèrie d'arxius que trobareu a la pàgina web següent: http://ume015.uab.es/llibre.

El primer arxiu és clock.vhd, que conté la descripció del mòdul *clock*. Aquest mòdul usa el rellotge de la placa per possibilitar que un polsador generi el senyal de rellotge del processador. Així podem fer una execució pas a pas (de flanc de rellotge a flanc de rellotge) del nostre codi de programa. Heu d'agafar el codi vhdl i fer el símbol d'aquest mòdul. Un exemple és el que es mostra a la Fig. 14.5 (els noms dels terminals d'entrada s'han de respectar a causa del que s'exposa a la secció 14.3).

A més també es volen visualitzar els registres que formen la UP. Per això us proporcionem el mòdul *lcd*, descrit en un arxiu vhdl que es diu lcd.vhd. D'aquest mòdul, n'heu

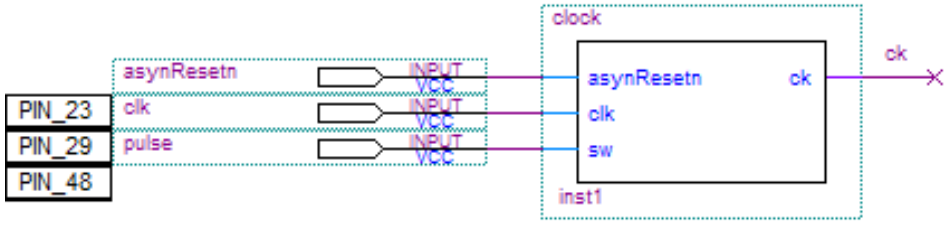


Fig.14.5 Modificació del rellotge del sistema (clock.vhd)

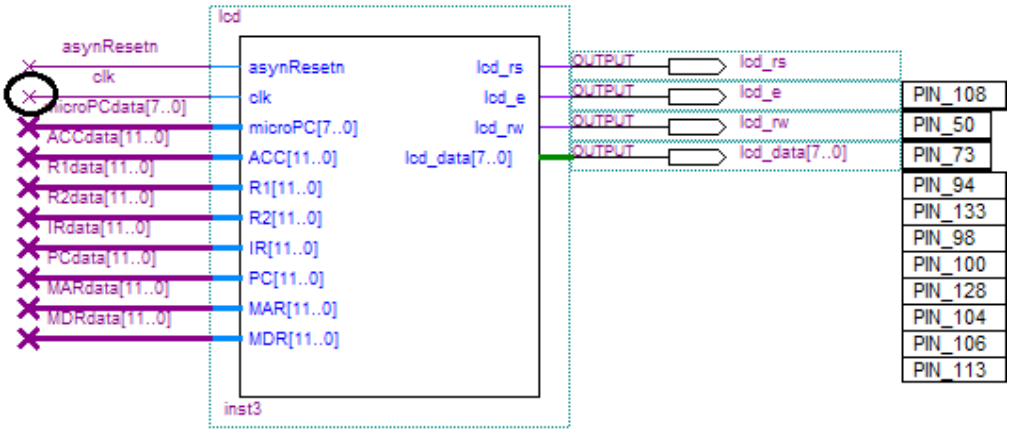


Fig. 14.6 Mòdul controlador de la pantalla LCD (lcd.vhd)

de generar el símbol i afegir-lo al circuit del processador, tal com es pot observar a la Fig. 14.6.

Heu de parar atenció al rellotge del mòdul *lcd* que no és el mateix que el que controla la UP i la UC. Aquest és el senyal clk (igual que el terminal d'entrada) i NO ck, que és el senyal que controla la UP i la UC. Penseu que clk és el rellotge del sistema i va a una freqüència de l'oscil·lador de 14,318 MHz, mentre que ara ck va a una freqüència determinada per les nostres pulsacions per tal de veure l'execució pas a pas.

Observeu que el pin 53 és el senyal *end* i el pin 54 és el senyal de sortida del rellotge ck. Això ho fem per visualitzar el senyal de finalització i el rellotge amb un led. Recordeu que heu de respectar els noms de les sortides.

Les connexions dels pins amb el display LCD estan disposades de manera que a l'hora de visualitzar el display el veureu com s'indica a la Fig. 14.7.

(mPC	ACC	R1	R2)
00	**000**	**000**	**000**
000	**000**	**000**	**000**
(IR	PC	MAR	MDR)

Fig. 14.7 Display de 16x2 caràcters

A la part de dalt es poden veure quatre registres. D'esquerra a dreta tenim els valors en format hexadecimal de mPC (registre microPC), ACC (l'acumulador), R1 (registre 1) i R2 (registre 2). A la fila de sota, i d'esquerra a dreta, es troben els valors en hexadecimal dels registres IR, PC, MAR i MDR.

D'altra banda, cal disposar del contingut de cadascun dels registres involucrats. Per això les sortides de la UP han de canviar. Partim d'una versió de la UP com la que es mostra a la Fig. 14.8 i s'ha de canviar per un mòdul on surten tots els busos de les sortides dels registres que són R1, R2, IR, PC, MAR, MDR i ACC. En el cas que es mostra a la Fig. 14.9 els noms serien: *R1data[11..0], R2data[11..0], IRdata[11..0], PCdata[11..0], MARdata[11..0], MDRdata[11..0]* i *ACCdata[11..0]*.

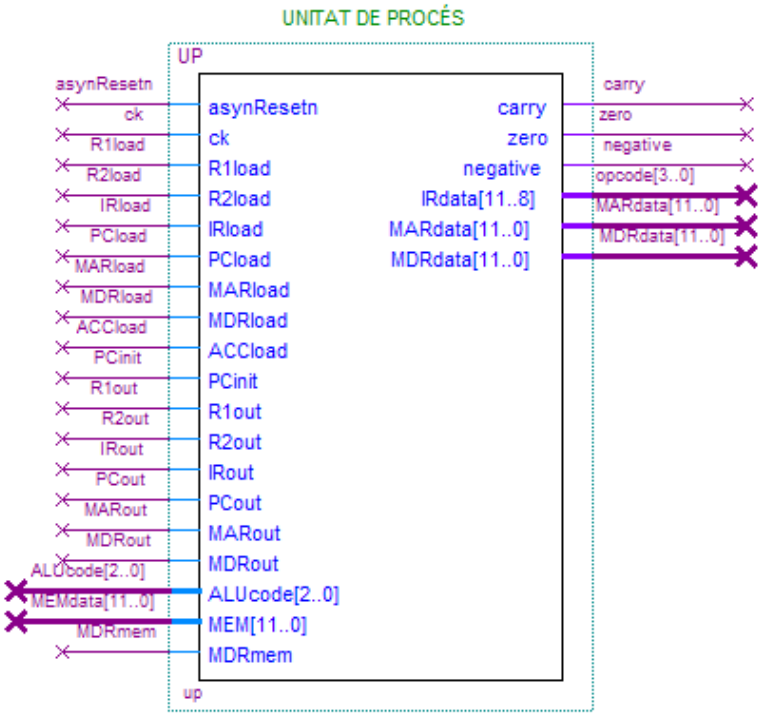


Fig. 14.8 Esquemàtic de la UP original

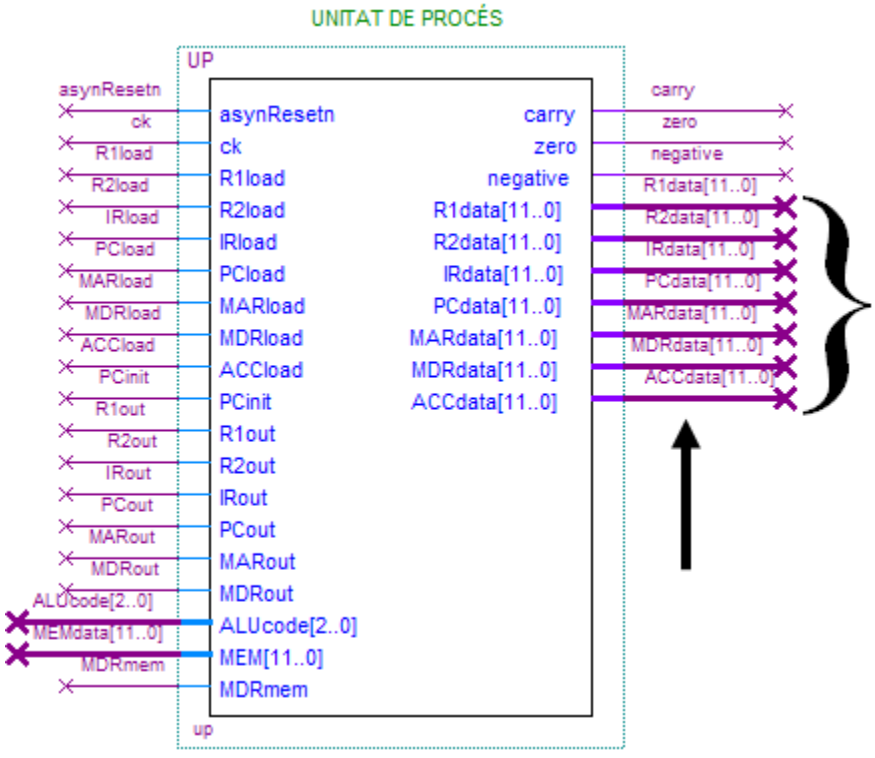


Fig. 14.9 Esquemàtic de la UP modificada

De la mateixa manera s'ha d'afegir un pin de sortida a la unitat de control amb el contingut del microPC (vegeu la Fig. 14.10). Això és necessari perquè el microPC es pugui veure en el *display* LCD.

MOLT IMPORTANT: Fixeu-vos que el senyal opcode[3..0] que connectava la sortida IRdata[11..8] de la UP i l'entrada opcode[3..0] de la UC passa a ser el senyal IRdata[11..8]. És a dir, ara l'entrada opcode[3..0] de la UC es connecta a IRdata[11..8]

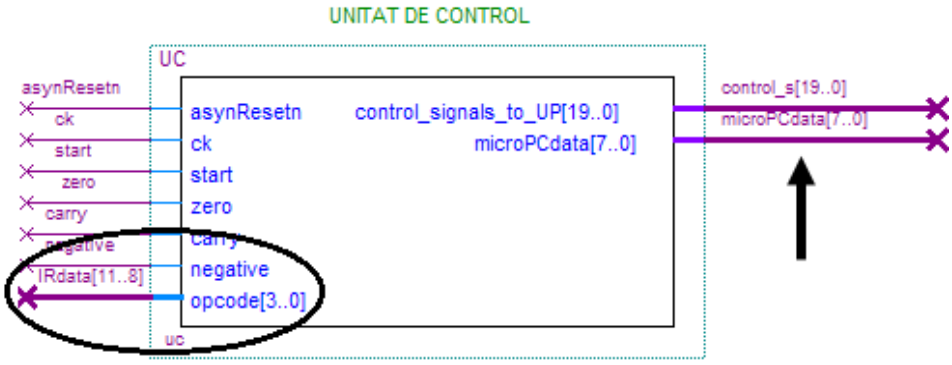


Fig. 14.10 UC modificada: pin de sortida afegit i nova connexió a l'entrada opcode

D'altra banda, el senyal d'entrada *start* s'ha de substituir pel nou senyal d'entrada *startn*. El senyal *start* es manté però passa a ser un senyal intern que equival a la negació del senyal d'entrada *startn*. Per això, entre l'entrada *startn* i el vostre senyal *start* heu de ficar un inversor, una cel·la not.

14.3. Associació de pins

Per fer l'associació dels pins d'entrada/sortida de la FPGA amb els terminals d'entrada/sortida del microprocessador, us proporcionem el fitxer pins.tcl. Aquest arxiu té tota la informació sobre les posicions dels pins i quina és la seva entrada/sortida vinculada, així com el dispositiu FPGA que s'utilitzarà (obriu l'arxiu per a més informació).

Per executar aquest arxiu heu d'anar a:

`Tools → Tcl Scripts...`

seleccioneu el fitxer de pins i premeu el botó **`Run`** (vegeu la Fig. 14.11).

(Aquest arxiu que us proporcionem hauria d'estar en el directori on teniu tot el muntatge del microprocessador).

Un cop realitzada l'assignació de pins, el circuit s'ha de compilar per obtenir l'arxiu de configuració adient per ser descarregat a la FPGA.

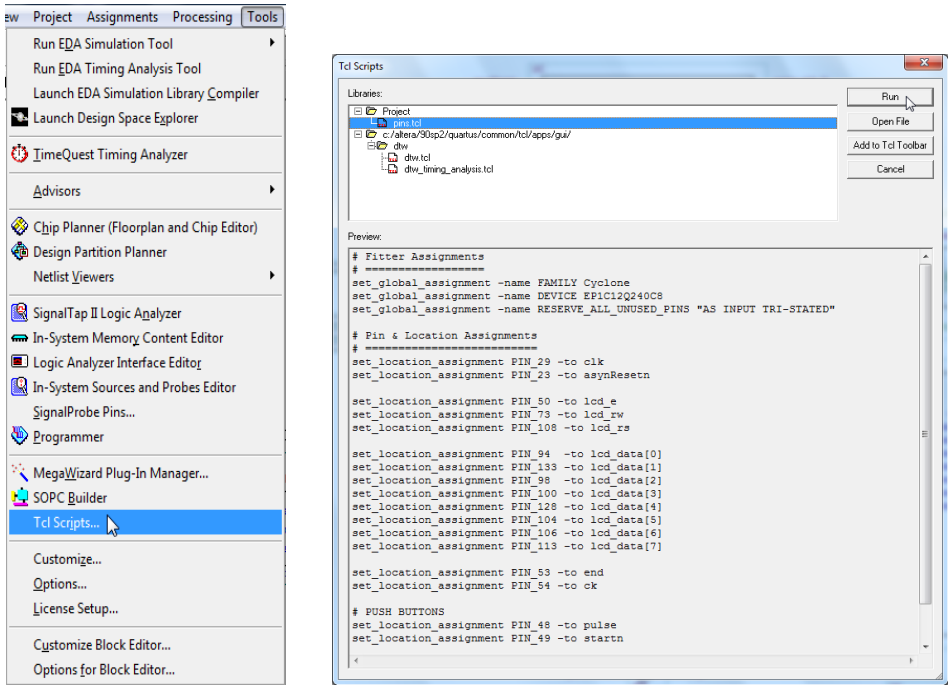


Fig. 14.11 Associació de pins

14.4. Programació

Una vegada ja teniu l'associació de pins amb el dispositiu FPGA es passa a fer la descàrrega de l'arxiu de configuració de la FPGA via el connector JTAG. Per això heu de connectar el connector JTAG a la clavilla adient. Hi ha dos connectors similars: heu d'escollir el que té l'etiqueta JTAG (en cas de dubte pregunteu al professor).

També heu de connectar la placa al transformador i aquest al corrent per alimentar la placa.

Finalment, cal programar la FPGA amb el nostre disseny, i la nostra assignació de pins. Per fer això cal pitjar sobre el botó *Programmer*.

Fig. 14.12 Programador

Un cop pitjat aquest botó s'obre una finestra com la que podeu veure a la figura següent. Si tot està connectat de manera correcta s'ha de veure que la connexió de la placa amb el PC es fa a través del *ByteBlasterII* (o *ByteBlasterMV)* que és el connector que se us ha proporcionat. El connector que té la placa és un JTAG

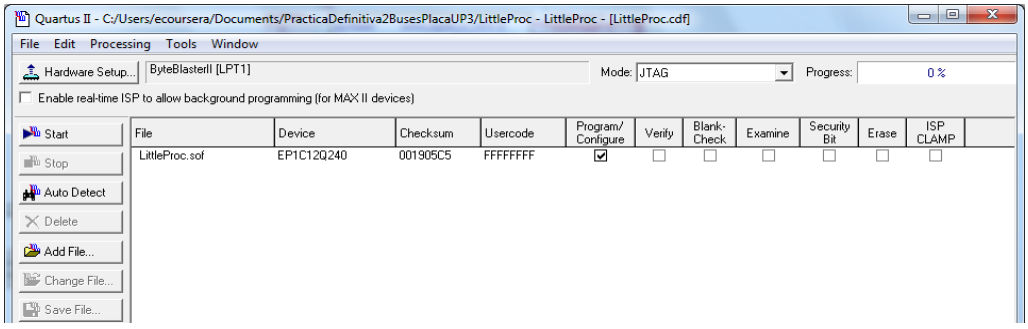


Fig. 14.13 Finestra de programació. La interfície de programació és ByteBlasterII

Si això no és així, és a dir, si no hi ha per defecte el *ByteBlaster* i el mode *JTAG*, s'ha d'anar a la pestanya de **`Hardware Setup`** i ajustar els paràmetres com es pot veure a la figura següent:

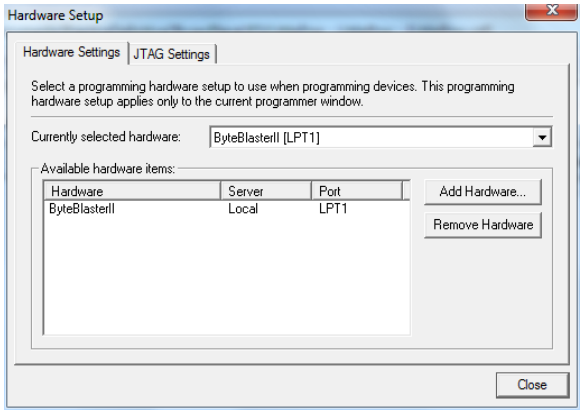


Fig. 14.14 ByteBlasterII mode JTAG

Heu de veure que en el *JTAG Settings* el *JTAG Server* és local i que està OK. A continuació, heu de triar el tipus de hardware i el port que és el LPT1 (port paral·lel). Per a això pitgeu el botó **`Add Hardware`**.

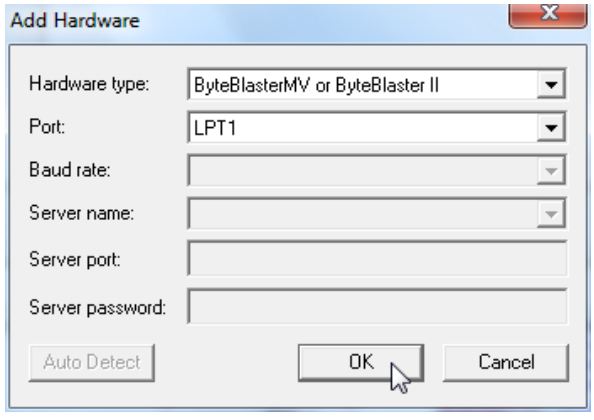


Fig. 14.15 Finestra per connectar port paral·lel

Un cop feta la inicialització del maquinari (**`Hardware Setup`**) heu de carregar el fitxer *<nom_fitxer>.sof* què és el que porta la informació de com s'han de connectar els CLB's (blocs lògics, assignació de pins d'entrada/sortida i memòria) dins de la FPGA per tal d'aconseguir la funcionalitat que heu simulat anteriorment. Aquest arxiu *.sof* es troba al directori de treball que esteu fent servir. S'ha de fixar també el dispositiu FPGA (que ja a hores d'ara hauríeu de saber quin és). Finalment, heu de seleccionar la casella de **`Program Configure`** i després pitjar el botó **`Start`**.

Recordeu:
File: <camí on està el vostre i triar l'arxiu .sof que voleu carregar a la FPGA>
Device: La FPGA que heu de fer servir (la Cyclone).
Program Configure: Sí (pitjar)

Llavors hi ha un procés de càrrega del fitxer *.sof* a la FPGA que pot trigar uns segons. Quan el *Progress* arriba al 100% la FPGA queda configurada i per tant ja hauria de tenir la funcionalitat que vosaltres voleu.

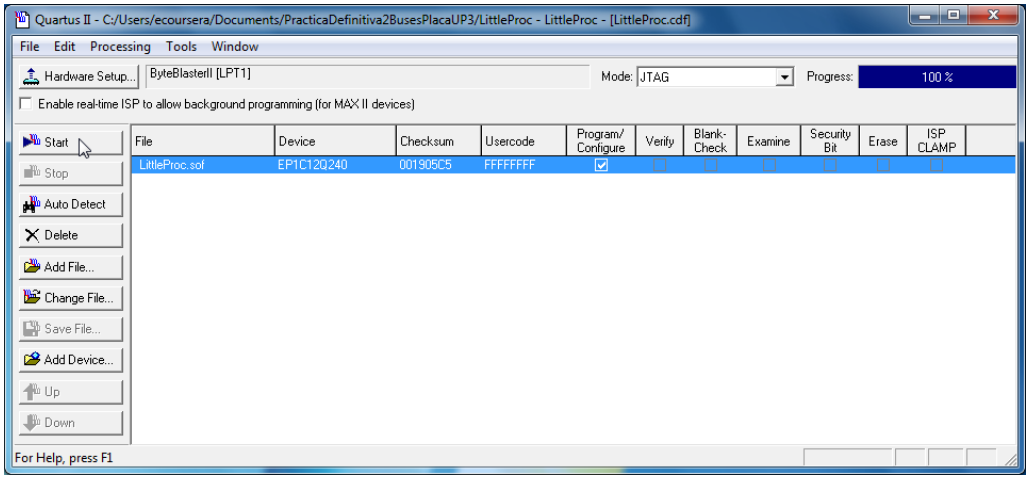


Fig. 14.16 Finestra per programar la placa: càrrega del fitxer .sof

15. Implementació a la placa Altera DE2

Implementar un circuit ja dissenyat a una FPGA és un procés que consta de diverses etapes. En primer lloc, resulta fonamental conèixer les característiques del model concret de FPGA que es farà servir. A més, l'FPGA habitualment està muntada sobre una placa conjuntament amb altres dispositius (Fig. 15.1). Aquesta placa té una sèrie d'interfícies per interactuar-hi, interfícies que poden ser d'entrada (per exemple, un port PS/2 per connectar-hi un teclat o un ratolí) o de sortida (per exemple, un port VGA per connectar-hi un monitor). També hi ha interfícies que tenen com a missió possibilitar la càrrega de la informació d'un disseny sobre el dispositiu FPGA. En conclusió, no solament s'han de conèixer les característiques de l'FPGA sinó també les de la placa on es troba.

L'FPGA és un xip que té una sèrie de pins d'entrada i sortida. En la placa aquests pins estan ja connectats a altres dispositius de la placa. Una etapa important és l'assignació de terminals d'entrades i sortides del nostre circuit als pins de l'FPGA. Per fer aquesta assignació s'ha de tenir en compte tant la funció concreta de les entrades i sortides del circuit com la manera específica com es vol interactuar amb elles des de l'exterior, interactuació que pot requerir de diversos dispositius dels que es troben a la placa.

En alguns casos pot ser necessari haver d'afegir algun subcircuit addicional o terminals al nostre disseny per facilitar la interactuació amb els dispositius de la placa que volem utilitzar. Per exemple, si la placa té un LCD i es vol treure informació a través d'aquest, molt probablement s'haurà d'afegir al circuit un mòdul que controli el funcionament de l'LCD i algunes sortides específiques per a aquest fi. El desenvolupament d'aquests mòduls addicionals i la seva integració amb el circuit dissenyat constitueixen també una altra etapa per la qual haurà de passar el dissenyador.

L'última etapa consisteix en el procés de programació de l'FPGA a través d'alguna de les interfícies que la placa té amb aquesta finalitat. Una vegada l'FPGA és programada, el circuit ja és una realitat i se'n pot comprovar el comportament.

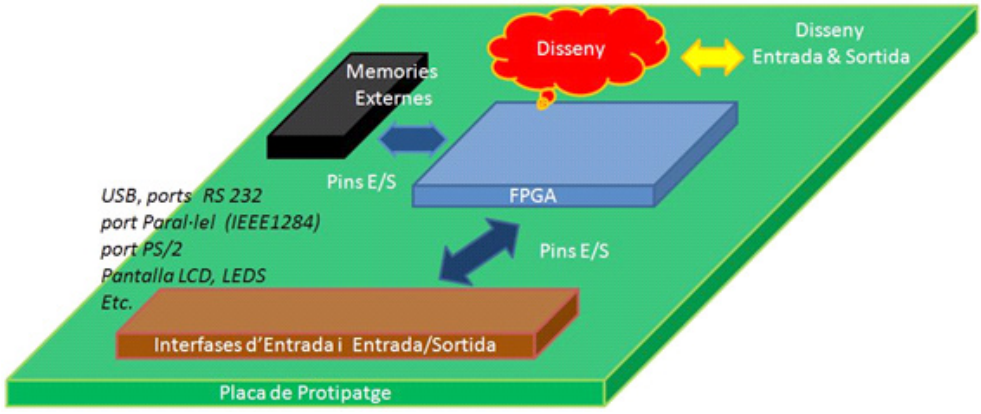


Fig. 15.1 Esquema bàsic d'una placa amb una FPGA

En aquest capítol es presenta com es fa la implementació de LittleProc en una FPGA de la família Cyclone II, concretament el model 2C35F672C6, que es troba a la placa de desenvolupament Altera DE2. En el primer punt del capítol es presenten les principals característiques de la placa Altera DE2. En el punt següent s'esquematitza el procés que cal seguir per realitzar la implementació de LittleProc, el qual està estructurat en tres fases. Finalment, els tres punts restants del capítol consisteixen en l'explicació de cadascuna d'aquestes fases.

15.1. Placa de desenvolupament Altera DE2

La placa DE2 té molts elements i característiques que permeten a l'usuari implementar un gran nombre de circuits, des de circuits elementals fins a circuits força complexos (per exemple, circuits multimèdia). La Fig. 15.2 mostra una imatge de la placa DE2. S'hi pot apreciar la disposició dels components principals i dels ports de connexió. A continuació, enumerem els elements més importants:

- *Un dispositiu FPGA Altera Cyclone II 2C35F672C6*
- *Un dispositiu de configuració sèrie Altera EPCS16*
- *Un element USB Blaster per a la programació de l'FPGA. Tant el mode de programació JTAG com el mode Active Serial són compatibles.*
- *512 KB de memòria SRAM*
- *8 MB de memòria SDRAM*
- *4 MB de memòria Flash*
- *Un sòcol per a targetes SD*
- *4 polsadors amb filtre antirebot*
- *18 commutadors*
- *Un mòdul LCD 16 x 2 (2 línies de 16 caràcters)*
- *8 displays de tipus 7-segments*
- *18 leds de color vermell*
- *9 leds de color verd*
- *Un rellotge de 50 MHz i un altre de 27 MHz*
- *Un connector per a un rellotge extern*
- *Un CODEC d'àudio de 24 bits (qualitat CD) amb connexions line-in, line-out i mic-in.*
- *Un DAC VGA (triple DAC d'alta velocitat de 10 bits) amb un connector VGA-out*
- *Un descodificador TV (NTSC/PAL) i un connector TV-in*
- *Un controlador Ethernet 10/100 amb un connector*
- *Un controlador Host/Slave USB amb connectors de tipus A i B*
- *Un transceptor RS-232 i un connector de 9 pins*
- *Un connector per a teclat o ratolí de tipus PS/2*
- *Un transceptor IrDA*
- *Dos capçals d'expansió de 40 pins amb protecció de díodes*
- *Un connector per a l'alimentació de la placa (9V) i un interruptor de encesa*

De tots aquests elements, per implementar LittleProc a la placa, s'utilitzaran principalment l'FPGA, dos polsadors, un commutador, la pantalla LCD, un *led* verd i un *led* vermell. El procés de programació de la placa es farà mitjançant l'element USB-Blaster i el seu port associat.

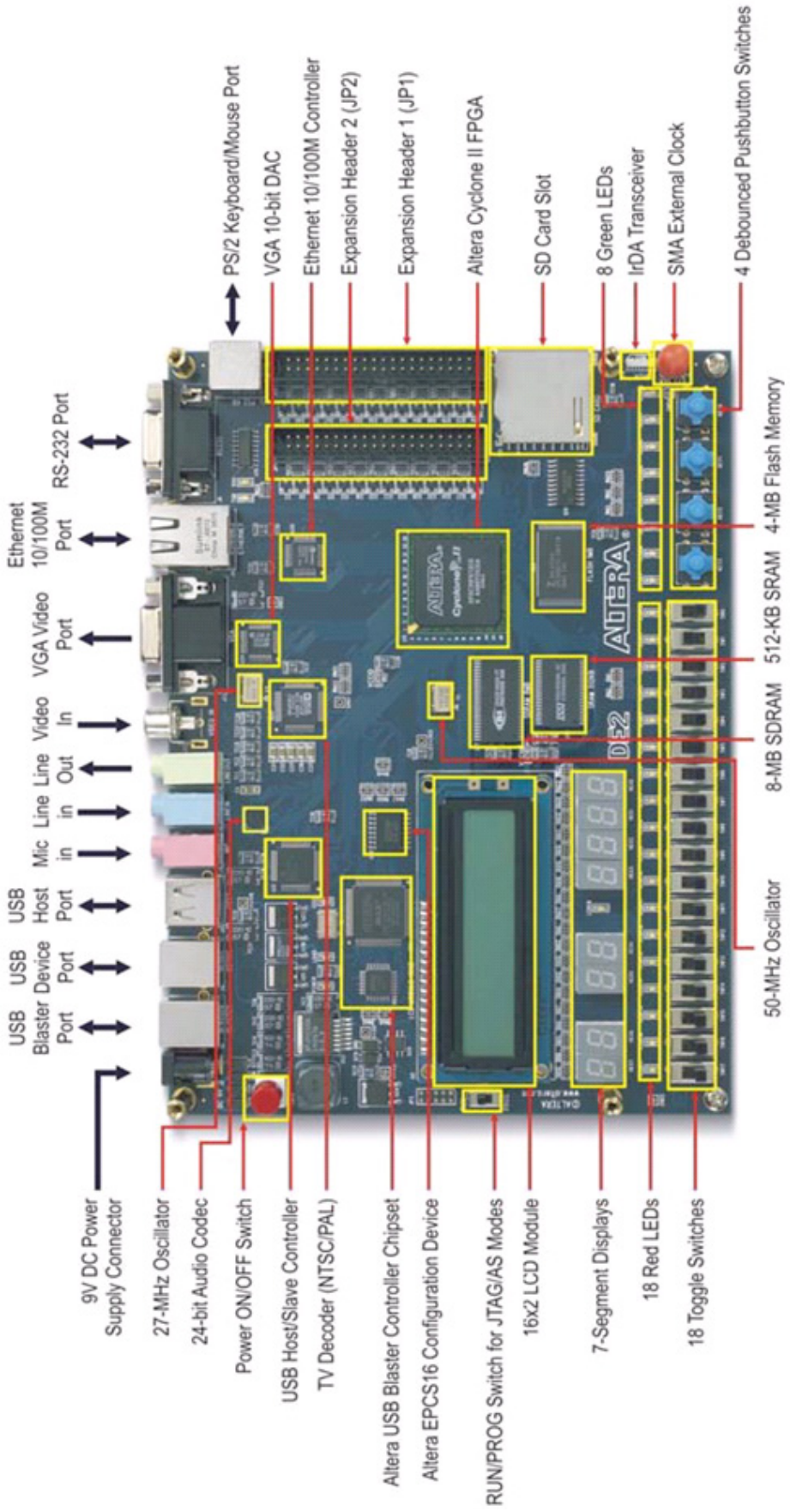


Fig. 15.2 Fotografia de la placa Altera DE2 (extreta del manual de la placa)

15.2. Implementació a placa

La implementació de LittleProc a la placa Altera DE2 està estructurada en tres fases:

1. Fer un seguit de modificacions en el circuit per possibilitar la visualització del funcionament de LittleProc. Aquestes modificacions consisteixen bàsicament en l'addició de dos mòduls i de diversos terminals d'entrada i sortida. Un mòdul permet la generació d'un rellotge «lent» (regit per un polsador) que controli el funcionament de LittleProc. L'altre mòdul possibilita veure el contingut de tots els registres de LittleProc a través de l'LCD de la placa.
2. Fer una associació de pins de l'FPGA a terminals de LittleProc que permeti la interacció amb LittleProc mitjançant un parell de polsadors, un commutador i l'LCD, principalment.
3. Realitzar la programació de l'FPGA mitjançant la interfície USB-Blaster.

En els punts següents es descriu amb un cert detall cadascuna de les tres fases esmentades. S'hi fa referència a diversos fitxers que es poden obtenir a la pàgina web http://ume015.uab.es/llibre.

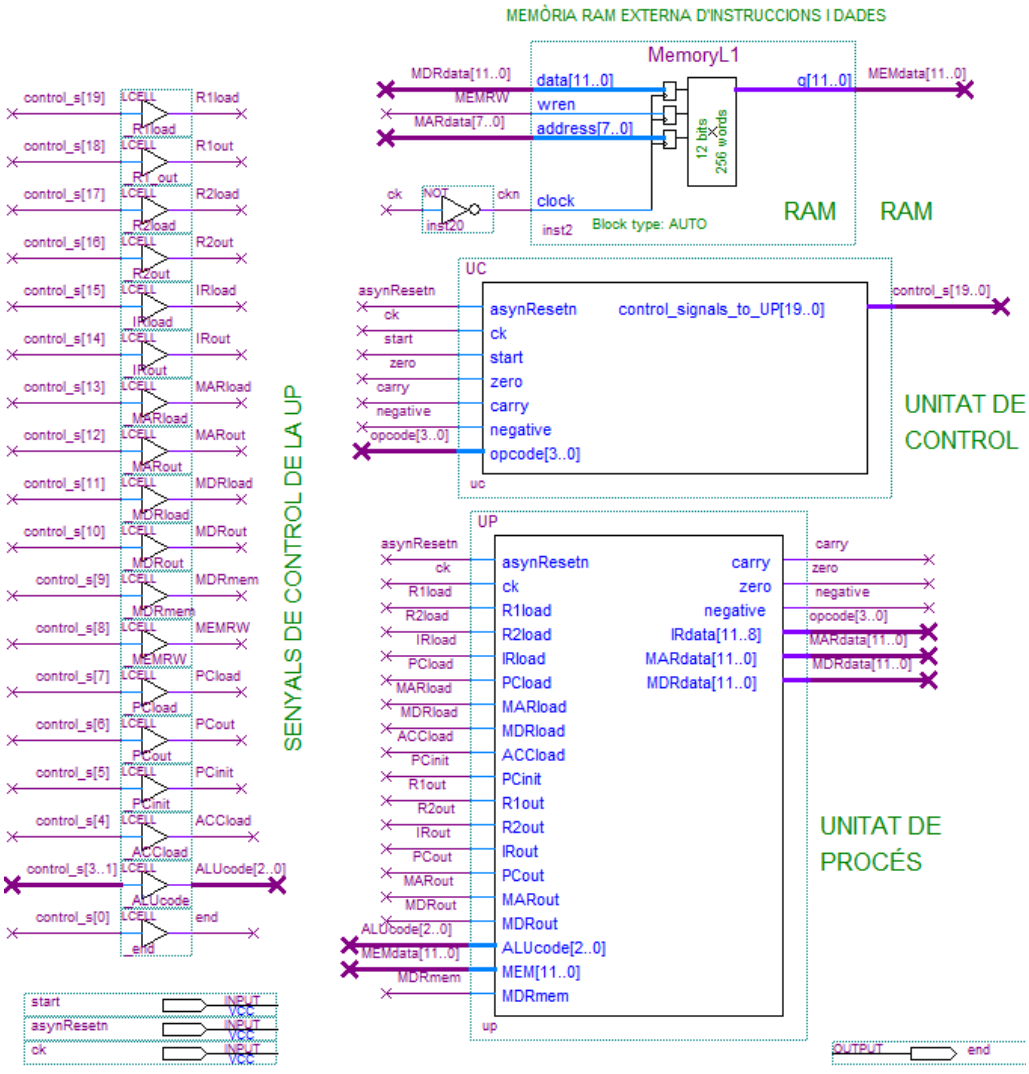


Fig. 15.3 LittleProc: circuit inicial

15.3. Modificacions de LittleProc

Les modificacions que s'han de fer a LittleProc tenen com a objectiu facilitar la seva monitorització i possibilitar-ne la interactuació a través de diversos dispositius de la placa. Es resumeixen en els punts següents:

- Afegir un mòdul que permeti aconseguir un rellotge «lent» mitjançant la realització manual de polsos amb un dels polsadors que hi ha a la placa. Dues entrades addicionals relacionades amb el mòdul s'afegiran al circuit.
- Possibilitar l'accés al contingut de tots els registres de la UP i de la UC. Això provocarà l'addició de pins de sortida tant a la UC com a la UP.
- Afegir un mòdul controlador de la pantalla LCD de la placa i els terminals de sortida requerits a tal fi.
- Possibilitar la visualització del rellotge del processador.

Les figures 15.3 i 15.4 mostren el circuit en el seu estat inicial i una vegada modificat, respectivament. En els punts següents es comenten les modificacions de manera detallada.

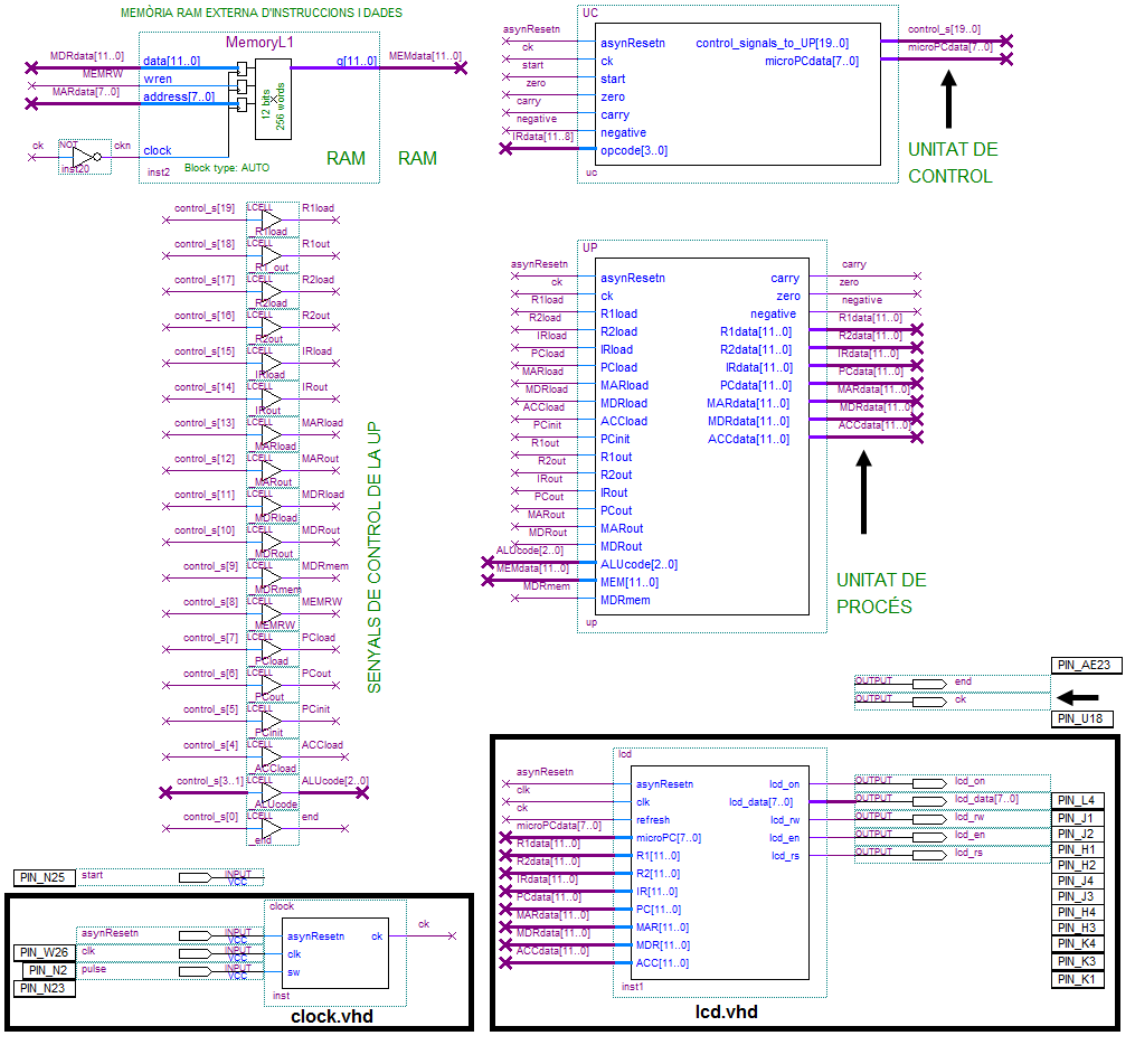


Fig. 15.4 LittleProc: circuit modificat

15.3.1. Generació d'un rellotge «lent»

Per aconseguir un rellotge «lent» fet a partir de polsos manuals, s'ha d'afegir un nou mòdul denominat *clock*. Aquest mòdul es troba descrit en VHDL al fitxer clock.vhd.

El mòdul *clock* utilitza un rellotge de la placa i el senyal procedent d'un polsador per generar un rellotge «lent» no periòdic regit per la realització de pulsacions manuals. Quan es prem el polsador i, a continuació, es deixa anar, es produeix un pols de rellotge simètric: ck val 0 durant mig segon, val 1 el mig segon següent i després torna a valdre 0. Aquest rellotge «lent» permetrà fer una execució controlada pas a pas (de flanc de rellotge a flanc de rellotge) del programa carregat a la memòria RAM.

Per incorporar el mòdul *clock*, s'obrirà en Quartus el fitxer clock.vhd i es crearà un símbol per al mòdul. Aquest símbol s'afegirà al circuit fent les connexions que es mostren a la Fig. 15.5. Es pot observar, que a més del mòdul, s'hauran d'agregar dos pins d'entrada: *clk* (rellotge de la placa) i *pulse* (senyal del polsador). El mòdul *clock* genera el senyal de rellotge *ck* que serà el rellotge amb el qual funcionarà el processador (per tant, s'ha d'eliminar del circuit el pin d'entrada *ck*, que ja no és necessari).

15.3.2. Accés al contingut de tots els registres

Per poder observar el comportament del processador resulta fonamental veure el contingut de tots els seus registres. Amb aquest fi es procedirà a modificar la UP i la UC afegint-hi terminals de sortida per treure a l'exterior el contingut de tots els registres. Analitzem, a continuació, cadascun dels dos casos.

La UP ja té els terminals de sortida *MARdata[11..0]* i *MDRdata[11..0]* que treuen el contingut dels registres MAR i MDR. A la UP s'afegiran els terminals de sortida *R1data[11..0]*, *R2data[11..0]*, *PCdata[11..0]* i *ACCdata[11..0]* per fer observable el contingut de R1, R2, PC i l'acumulador, respectivament. A més, el terminal de sortida *IRdata[11..8]* que treu els quatre primers bits del registre IR serà substituït pel terminal de sortida *IRdata[11..0]*, que permetrà observar el contingut de tot l'IR. D'aquesta manera s'aconsegueix que el contingut de tots els registres de la UP siguin accessibles des de l'exterior de la UP. Les figures 15.6 i 15.7 mostren el símbol i la connectivitat de la UP inicial i els de després de fer les modificacions, respectivament. A més dels canvis en els terminals, es pot observar com el senyal *opcode[3..0]* ha estat substituït pel senyal *IRdata[11..0]*. Aquest canvi tindrà repercussió en la connectivitat de la UC, com veurem a continuació.

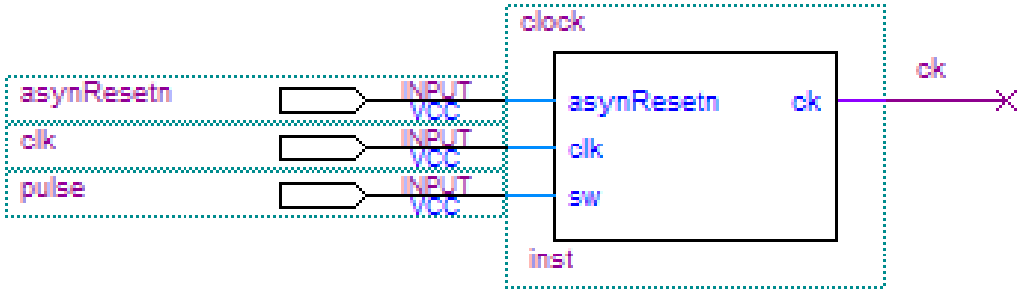


Fig. 15.5 Modificació del rellotge del sistema: addició del mòdul *clock*

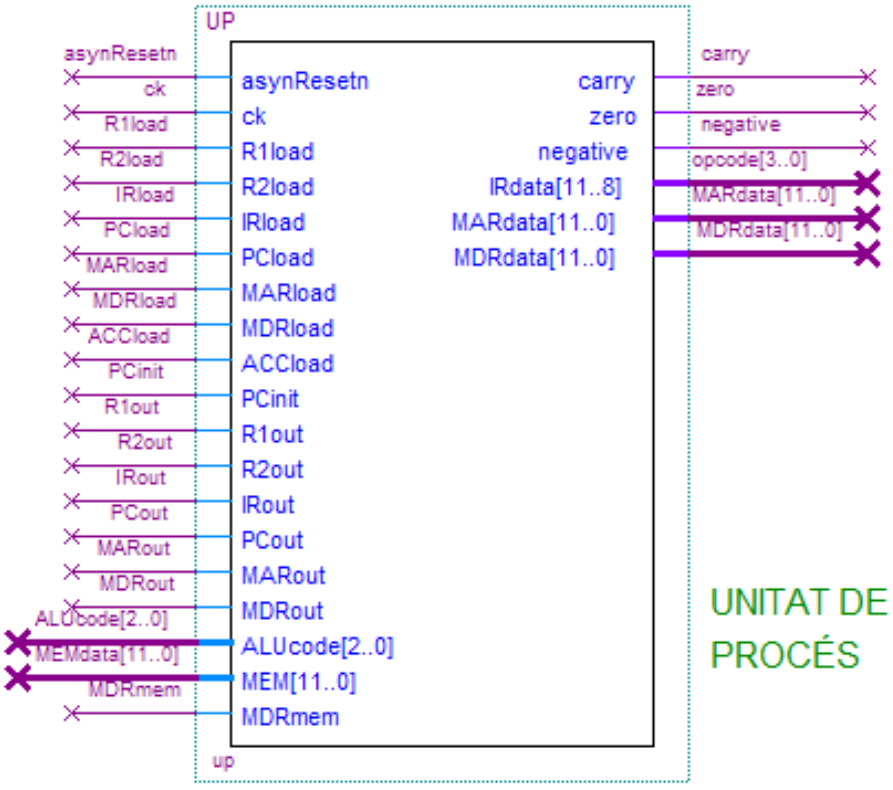


Fig. 15.6 Unitat de procés: símbol i connectivitat inicials

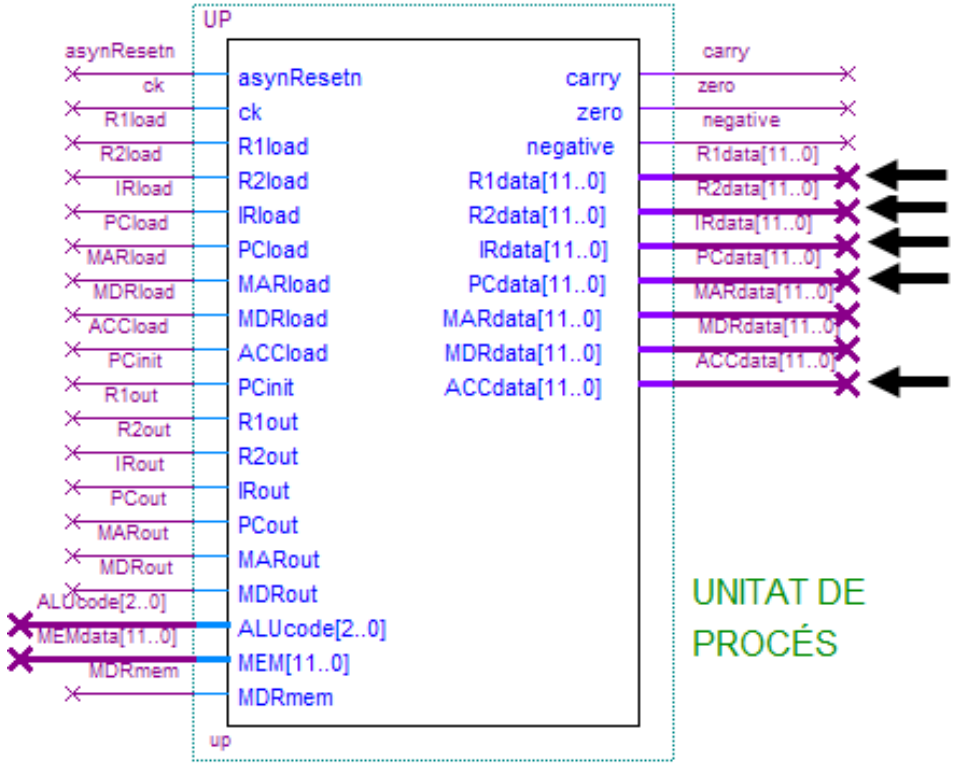


Fig. 15.7 Unitat de procés: símbol i connectivitat modificats

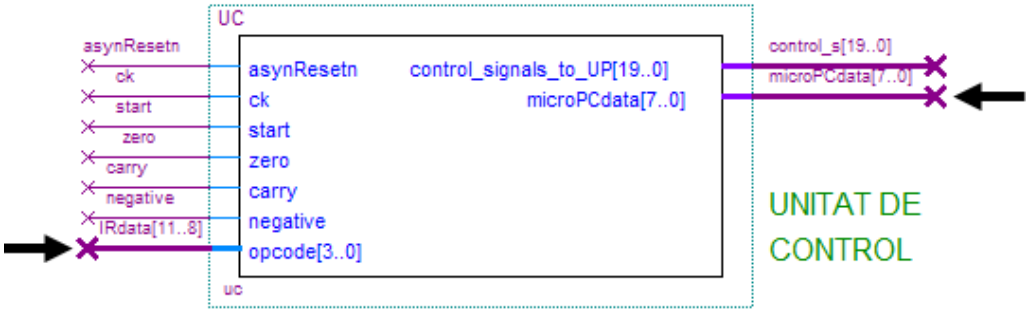


Fig. 15.8 Unitat de control: símbol i connectivitat modificats

En el cas de la UC, s'afegirà el terminal de sortida *microPCdata[7..0]* per possibilitar l'observació del contingut del registre microPC. La figura 15.8 mostra el símbol i la connectivitat de la UC després de fer les modificacions. És interessant veure que l'entrada *opcode[3..0]* abans estava connectada al senyal anomenat *opcode[3..0]* i ara està connectada al senyal *IRdata[11..8]*. Aquest canvi és conseqüència directa del canvi de la connectivitat de la UP que esmentaven al final del paràgraf anterior.

15.3.3. Ús de l'LCD

Observar el comportament del processador requereix visualitzar el contingut dels seus registres, la qual cosa es farà utilitzant l'LCD que hi ha a la placa. A tal fi s'ha d'afegir un nou mòdul denominat *lcd*. Aquest mòdul es troba descrit en VHDL al fitxer *lcd.vhd*.

El mòdul *lcd* és el controlador de l'LCD i determina què es mostra per pantalla. Aquest mòdul és un circuit seqüencial regit pel senyal *clk* (rellotge de la placa). També utilitza el senyal *ck* (rellotge del processador) per saber quan ha de refrescar el contingut de la pantalla. Cada vegada que el senyal *ck* experimenti un flanc de pujada, el mòdul *lcd*, després d'un petit període d'espera, refrescarà la pantalla. A la pantalla de l'LCD es mostra el contingut en format hexadecimal dels 8 registres que formen part del processador (vegeu Fig. 15.9). A la primera línea es presenten els valors dels registres microPC, ACC, R1 i R2. A la segona línia es troben els valors dels registres IR, PC, MAR i MDR.

Per incorporar el mòdul *lcd*, s'obrirà en Quartus el fitxer *lcd.vhd* i es crearà un símbol per al mòdul. Aquest símbol s'afegirà al circuit fent les connexions que es mostren a la Fig. 15.10. Es pot observar, que el mòdul rep com a entrades, a més del senyals ja esmentats, els busos amb el contingut de cada registre. Quant a les seves sortides, aquestes son les encarregades de controlar directament l'LCD i per a cadascuna d'elles, s'haurà

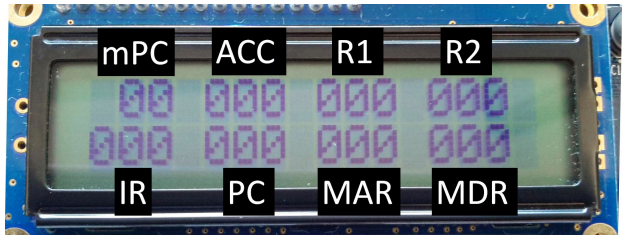


Fig. 15.9 LCD: presentació del contingut dels registres

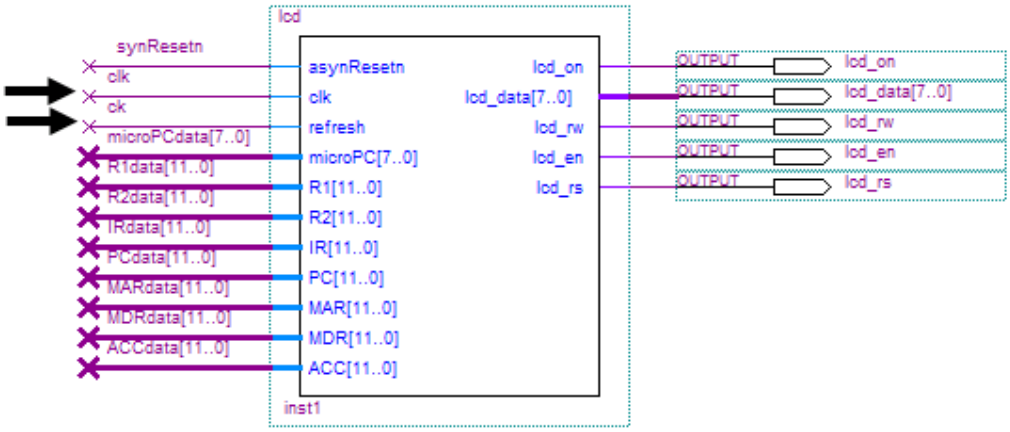


Fig. 15.10 Mòdul controlador de l'LCD

d'afegir un pin de sortida al circuit. No comentarem la funcionalitat concreta d'aquestes sortides, ja que és un tema que queda fora de l'abast d'aquest llibre.

L'LCD no solament serveix per mostrar el contingut dels registres del processador sinó també per fer palesa l'activació del senyal *asynResetn*. Quan s'activa *asynResetn* mitjançant un polsador, la pantalla de l'LCD s'apaga. Al contrari, quan *asynResetn* està desactivat, la pantalla està encesa i mostra el contingut dels registres.

15.3.4. Visualització del rellotge del processador

La visualització del senyal *ck*, el rellotge del processador generat pel mòdul *clock*, es farà mitjançant un *led* verd. Per fer possible això, s'ha d'afegir un terminal de sortida al circuit per treure el senyal de rellotge *ck*. Aquest terminal addicional de sortida es pot apreciar en la Fig. 15.4 on es mostrava el circuit global modificat.

Quan el senyal *ck* valgui 1 el *led* verd s'encendrà. Cada vegada que es prem i es deixa anar el polsador que regeix el rellotge del processador, després de mig segon, s'encén el *led* verd i es manté encès durant mig segon.

15.4. Associació de pins

Per fer l'associació dels terminals de LittleProc amb pins d'entrada/sortida de l'FPGA, s'haurà d'utilitzar el fitxer pins.tcl. Aquest arxiu conté informació on s'indica el model de FPGA i, per a cada terminal de LittleProc, quin pin de l'FPGA té associat. L'arxiu pins.tcl és un arxiu de tipus textual que podeu obrir si esteu interessats en veure tota la informació que conté. Mitjançant aquesta associació de pins es vincula els terminals de LittleProc amb certs dispositius de la placa de la manera següent:

- l'entrada *asynResetn* amb el polsador KEY3
- l'entrada *clk* amb el rellotge de la placa de 50 MHz
- l'entrada *pulse* amb el polsador KEY1
- l'entrada *start* amb el commutador SW0
- la sortida *end* amb el *led* vermell LEDR0

- la sortida *ck* amb el *led* verd LEDG4
- les sortides *lcd_on*, *lcd_data[7..0]*, *lcd_rw*, *lcd_en*, *lcd_rs* amb l'LCD

La interactuació amb LittleProc la fareu a través dels dispositius de la placa esmentats en la relació anterior.

Per fer l'associació de pins haureu de gravar el fitxer pins.tcl al directori del projecte de Quartus, on teniu tot el muntatge del processador, i executar l'ordre:

`Tools → Tcl Scripts...`

Aleshores seleccioneu el fitxer pins.tcl i premeu el botó **`Run`** (vegeu la Fig. 15.11).

Un cop realitzada l'assignació de pins, el circuit s'ha de compilar per obtenir l'arxiu de configuració (fitxer .sof), que serà descarregat a l'FPGA a l'etapa de programació.

15.5 Programació

Una vegada heu fet l'associació de pins, ja podeu fer la programació de l'FPGA mitjançant la descàrrega sobre aquesta del fitxer .sof generat a l'etapa anterior, a través del connector USB-Blaster.

En primer lloc, heu d'alimentar la placa connectant-la al transformador i aquest al corrent elèctric. A continuació connecteu l'ordinador amb la placa mitjançant el cable USB. Aquest cable s'ha de connectar al connector USB-Blaster de la placa (no us confongueu amb el connector USB-Device que es troba just al costat). A continuació, comproveu que l'interruptor del mode JTAG (es troba al mig del lateral esquerre) està en

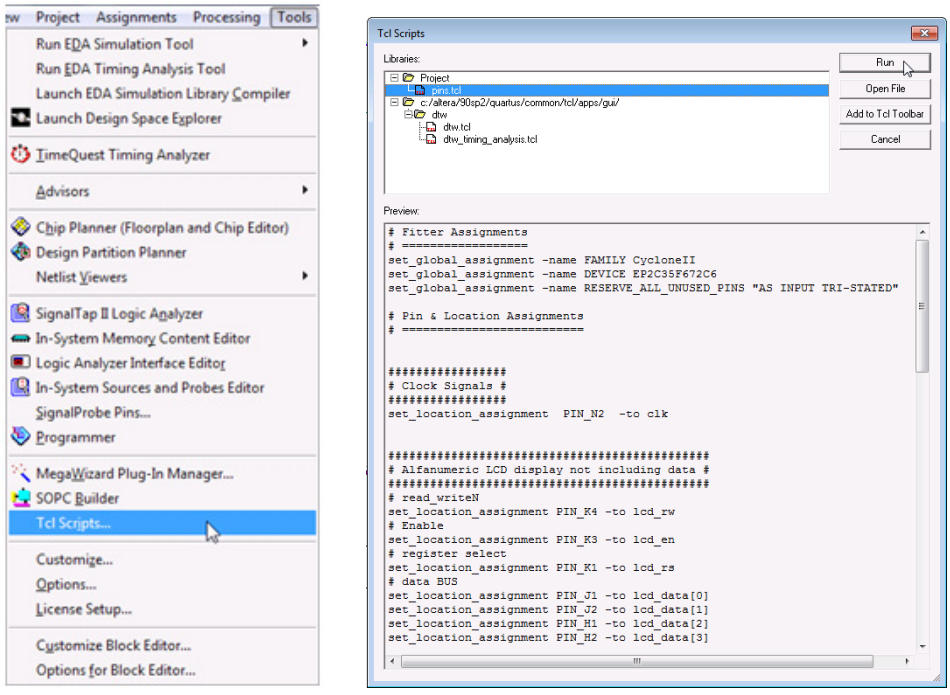


Fig. 15.11 Associació de pins

posició **RUN**. Finalment, per posar en funcionament la placa heu de prémer l'interruptor vermell que es troba a prop del cantó superior esquerre.

Ara ja es pot procedir a programar la placa amb el disseny de LittleProc i l'assignació de pins establerta. Per fer això, cal pitjar el botó **Programmer** (Fig. 15.12).

Com a conseqüència d'aquesta ordre, s'obre una finestra com la que es mostra a la Fig. 15.13. En aquesta finestra ha d'especificar com a *Mode*, **JTAG**, i a l'apartat *Hardware Setup* ha d'indicar **USB-Blaster**.

En cas que a l'apartat *Hardware Setup* hagi un valor diferent, s'ha de prémer el botó **Hardware Setup** i, a la finestra que s'obre, especificar els paràmetres d'acord amb la Fig. 15.14. Feu servir el botó **Add Hardware** per afegir la interfície USB-Blaster i el camp **Currently selected hardware** per seleccionar-la.

Fig. 15.12 Botó per a la programació de la placa

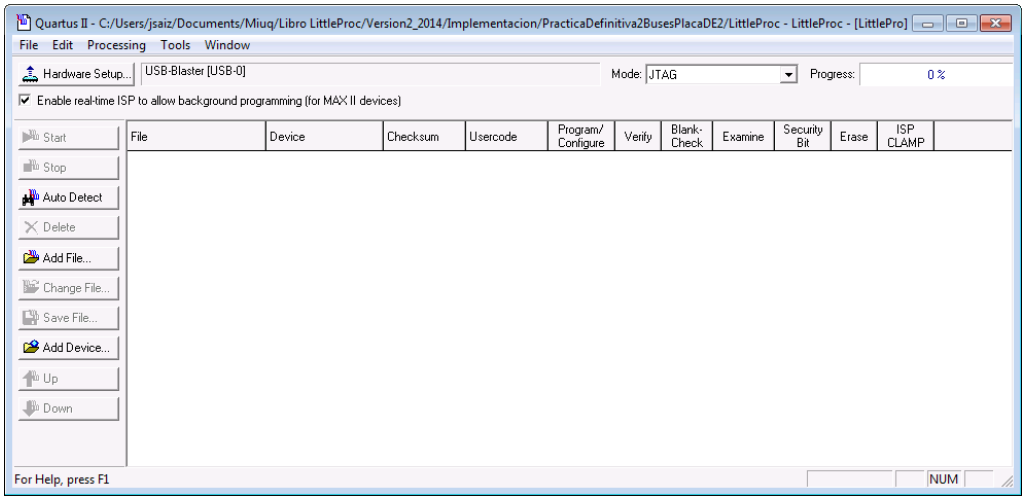


Fig. 15.13 Finestra de programació. La interfície de programació és USB-Blaster

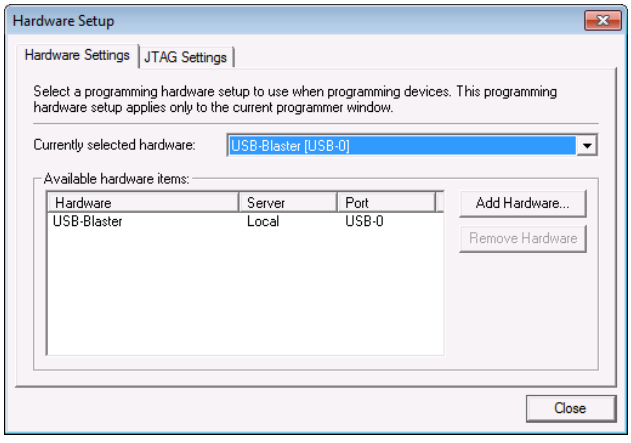


Fig. 15.14 Especificació de la interfície de programació USB-Blaster

Un cop especificat USB-Blaster com a interfície de programació heu d'indicar a l'apartat *File* el fitxer **`LittleProc.sof`**, que és el que porta tota la informació de com s'han d'interconnectar els elements (bloc lògics, pins de l'FPGA i memòria) dins de l'FPGA per aconseguir la implementació de LittleProc. Si l'apartat *File* ja conté un fitxer .sof, aquest se seleccionarà i es premerà el botó **`Delete`** per esborrar-lo. Per especificar el fitxer .sof del vostre disseny premeu el botó **`Add File`** i seleccioneu el fitxer **`LittleProc.sof`** (recordeu que es troba al directori del projecte). A més, heu de marcar la casella **`Program/Configure`**. Finalment, per concloure el procés de programació i realitzar la càrrega del fitxer .sof, heu de prémer el botó **`Start`** (vegeu Fig. 15.15).

El procés de càrrega del fitxer .sof triga uns quants segons, durant els quals el *led* blau LOAD de la placa s'encén. En finalitzar la càrrega, aquest *led* s'apaga. Si tot el procés s'ha fet de manera correcta, LittleProc es troba ja implementat a l'FPGA de la placa: LittleProc és ja una realitat!

A més, a la pantalla de l'LCD s'hauria de veure el contingut inicial de tots els registres, el qual és zero.

A partir d'aquest moment, mitjançant el polsador que regeix el rellotge del processador i el commutador vinculat al senyal *start* de LittleProc, podeu comprovar el correcte funcionament de LittleProc observant com varia el contingut dels seus registres a través de l'LCD.

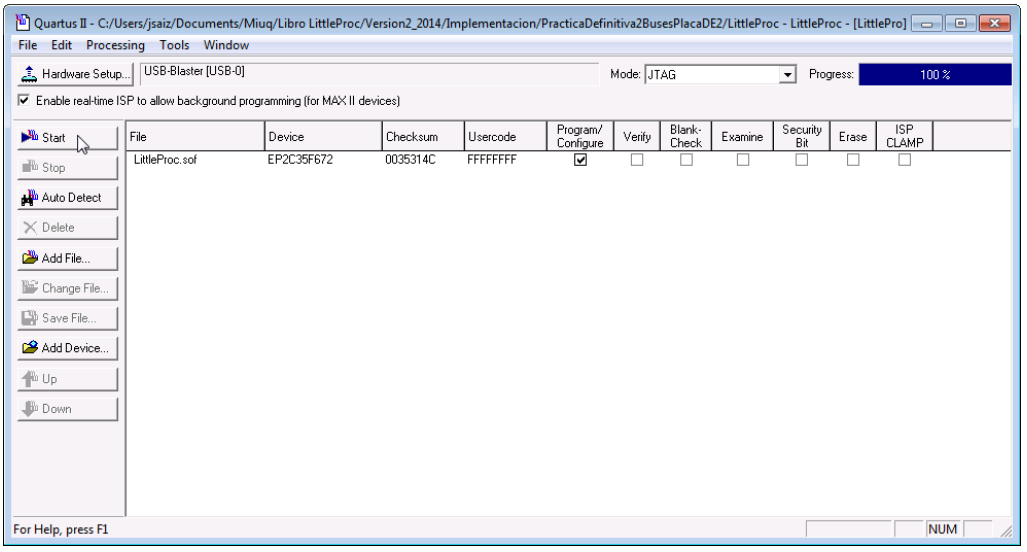


Fig. 15.15 Càrrega del fitxer LittleProc.sof

16. Col·lecció de problemes

16.1. Qüestionari escrit – Capítols 1 - 5

1. Expliqueu què són el *grid size* i l'*end time*. Quines són les diferències principals entre una simulació funcional i una *timing simulation*? Com sap el simulador de Quartus II quin és el fitxer d'ones que ha d'usar per fer la simulació?

2. De les afirmacions següents digueu quines són certes i quines són falses, i justifiqueu la resposta.
   a) Els resultats d'una simulació es poden veure quan s'executa la simulació però no es poden emmagatzemar per veure'ls posteriorment.
   b) Si es fa una simulació i, a continuació, es vol fer una nova simulació amb altres estímuls d'entrada, no cal recompilar el projecte.
   c) Suposem que tenim un disseny amb la jerarquia de cel·les de la Fig. 16.1. Si fem aquest disseny amb Quartus II, la *top level entity* serà la cel·la F.
   d) Suposem que s'acaba de simular una cel·la i que a continuació es vol simular una altra cel·la diferent (no inclosa en cap cel·la de jerarquia superior). Aleshores cal definir prèviament la cel·la a simular com la *top level entity* del projecte però no és necessari recompilar el projecte.

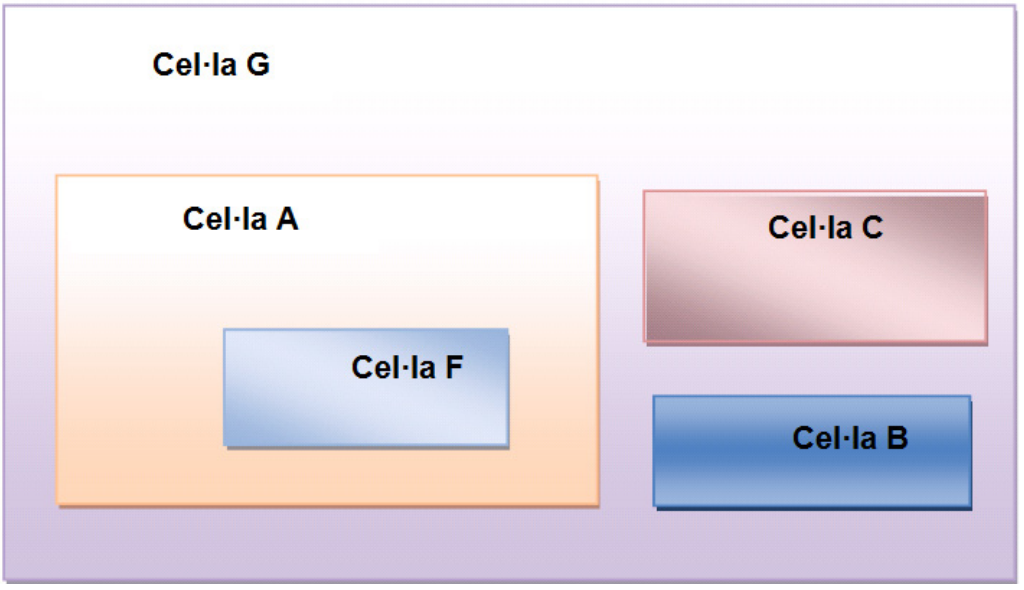


Fig. 16.1 Jerarquia d'un circuit

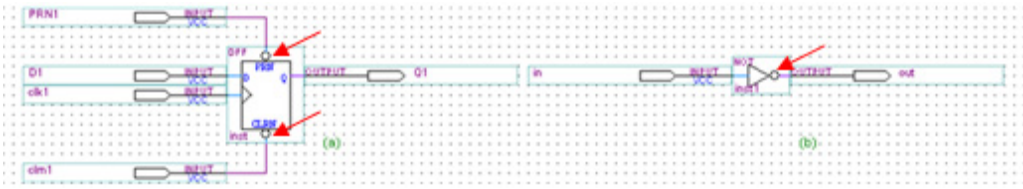


Fig. 16.2 Significat dels cercles en els símbols

3. Amb quina ordre Quartus II permet determinar el temps de retard d'un circuit combinacional? Presenta la cel·la ctrlSeq el mateix retard des de qualsevol entrada fins a qualsevol sortida? Quin és el retard màxim que presenta la cel·la ctrlSeq? Feu una captura de pantalla corresponent a l'execució de l'esmentada ordre que acrediti les vostres respostes relatives als retards.

4. En els mòduls (a) i (b) de la Fig. 16.2 observeu que hi ha una ○ (bola). Què significa aquest símbol en l'esquema (a)? I en l'esquema (b)?

5. Feu una simulació amb retards on es vegi el funcionament del registre de baldes. En aquesta simulació el senyal de control de les baldes ha de començar valent 0. A més, hi ha d'haver algun cas en què l'entrada de dades canviï quan el senyal de control valgui 1 i un altre en què canviï quan el senyal de control valgui 0. Expliqueu detalladament el comportament del registre indicant si aquest funciona per flanc o per nivell.

6. Feu una simulació amb retards de la cel·la PC en la qual es vegi:
   — una situació d'inicialització
   — una situació de càrrega de dades procedent de l'entrada *datain*
   — una situació de *reset*
   — el manteniment de la dada emmagatzemada en el registre en el període anterior de rellotge

   Funciona el registre PC per flanc o per nivell? Justifiqueu-ho. Exceptuant l'entrada de rellotge, quines de les seves entrades són asíncrones? I síncrones? I actives a alta? I actives a baixa?

7. Es podria simplificar la cel·la ctrlSeq? En cas afirmatiu expliqueu com.

8. Quants *flip-flops* té la cel·la microPC? I baldes (*latches*)? Quina és la taula de veritat d'un *buffer*? I la d'un buffer *tristate*?

9. Expliqueu detalladament com s'aconsegueix la inicialització de la cel·la PC amb el valor corresponent al vostre nombre de grup.

16.2. Qüestionari escrit – Capítol 6

1. Modifiqueu el codi de l'incrementador perquè quan rebi el valor 255 doni com a resultat 255, és a dir, no incrementi. Mostreu el codi i captureu una simulació que demostri que l'incrementador per valors inferiors a 255 incrementa i que per al 255, manté el mateix valor.

2. Feu un registre en VHDL de característiques similars al registre *reg* però que funcioni per nivell en lloc de funcionar per flanc. Mostreu el codi i captureu una simulació que demostri el seu funcionament (la simulació ha d'incloure tots els casos representatius del seu funcionament).

3. Feu el registre PC (desenvolupat mitjançant captura d'esquemes durant la segona sessió de pràctiques) en VHDL. Mostreu el codi i captureu una simulació que demostri el seu funcionament (la simulació ha d'incloure tots els casos representatius del seu funcionament).

4. De les sortides de l'ALU (C, *zero*, *negative* i *carry*), quines estan regides pel senyal de rellotge?

5. Expliqueu breument la manera òptima d'implementar una divisió per 4 d'un nombre signat, de manera combinacional a l'ALU.

6. Escriviu el codi que correspon al primer procés de l'ALU.

7. Escriviu el codi que correspon al segon procés de l'ALU.

8. Captureu una simulació en la qual es vegi clarament que el senyal de zero no es veu modificat per la realització de l'operació d'incrementació.

9. Captureu una simulació en la qual es vegi clarament la realització de tres sumes de nombres signats:
   — un nombre negatiu i un de positiu (aquest últim de magnitud major)
   — un nombre negatiu i un de positiu (aquest últim de magnitud menor)
   — el nombre positiu més gran i el valor 6.
   Expliqueu els resultats (incloent-hi els senyals de condició) que es produeixen.

10. Captureu una simulació en la qual es vegi clarament la realització d'una resta on es resten dos nombres iguals. Expliqueu els resultats (incloent-hi els senyals de condició) que es produeixen.

11. Captureu una simulació en la qual es vegi clarament la realització de dues operacions de desplaçament, l'una amb el *carry* activat i l'altra amb el *carry* desactivat. Expliqueu què passa a la simulació.

12. Expliqueu molt breument per a què serveix *carryV*.

13. Quins senyals formen part de la llista de sensibilitat de l'IRdecoder? Per què?

16.3. Qüestionari escrit – Capítols 7 - 13

1. Ompliu la taula de la ROM afegint el microcodi del cicle d'execució de les instruccions que falten. Per facilitar la tasca de correcció poseu cada microinstrucció com una línia de 24 dígits binaris sense espais entre ells.

Exemple:
-- LDR1
001000000110000000000110
001000000000000000000000
001000000000101000000000
001010000000010000000000
000100000010000000000000

2. Desenvolupeu una de les instruccions següents i mostreu el microcodi que implementa el seu cicle d'execució, així com una simulació demostrant-ne l'execució correcta.

Opcode	Instrucció	Paràmetres	Comentaris
11	DJNZR1 EXT	Posició de memòria	*Decrement R1 and Jump when No Zero*. Decrementa R1 i si R1 és diferent de 0 aleshores salta a la posició de memòria indicada. Indicació: per implementar el decrementador (operació no inclosa a l'ALU), heu de generar primerament el valor de 0 (p. ex., restant R1 amb R1) i posteriorment incrementar aquest 0 amb 1. Ara ja podeu restar a R1 aquest 1. Per fer aquest procés podeu fer servir el registres MAR i MDR com a registres temporals..
11	DJNZR2 EXT	Posició de memòria	*Decrement R2 and Jump when No Zero*. Decrementa R2 i si R2 és diferent de 0 aleshores salta a la posició de memòria indicada. Indicació: per implementar el decrementador (operació no inclosa a l'ALU), heu de generar primerament el valor de 0 (p. ex., restant R2 amb R2) i posteriorment incrementar aquest 0 amb 1. Ara ja podeu restar a R2 aquest 1. Per fer aquest procés podeu fer servir el registres MAR i MDR com a registres temporals..
12	DUP EXT	Posició de memòria	Duplica el valor emmagatzemat a la posició de memòria indicada i guarda el resultat de la duplicació a la mateixa posició de memòria.
12	TRI EXT	Posició de memòria	Triplica el valor emmagatzemat a la posició de memòria indicada i guarda el resultat de la triplicació a la mateixa posició de memòria.

13	DELAY	Cap	Decrementa el valor que hi hagi a R1 fins que arribi a 0. Indicació: per implementar el decrementador (operació no inclosa a l'ALU), heu de generar primerament el valor de 0 (p. ex., restant R1 amb R1) i posteriorment incrementar aquest 0 amb 1. Ara ja podeu restar a R1 aquest 1. Per fer aquest procés podeu fer servir el registres MAR i MDR com a registres temporals. Aquesta decrementació s'ha de realitzar fins que el flag de zero s'activi.
13	NEG EXT	Posició de memòria	Calcula el negatiu del valor emmagatzemat a la posició de memòria indicada i guarda el resultat de l'operació a la mateixa posició de memòria.
14	ROLR1 N	Nombre de bits	Fa la rotació a l'esquerra de N bits en R1. Indicació: cal fer servir la instrucció en VHDL *rol*. Atès que el format estàndard de les dades és l'stdlogicvector i la funció *rol* necessita les dades amb un altre format, cal fer el *casting* de les dades. En concret, el prototipus o format E/S de la funció seria aquest: *bitvector (R1) <= (bitvector de R1) rol (integer de N)*
14	RORR1 N	Nombre de bits	Fa la rotació a la dreta de N bits en R1. Indicació: per fer la rotació a la dreta de R1, cal fer servir la instrucció en VHDL *ror*. Atès que el format estàndard de les dades és l'stdlogicvector, i la funció *ror* necessita les dades amb un altre format, cal fer el *casting* de les dades. En concret, el prototipus o format E/S de la funció seria aquest: *bitvector (R1) <= (bitvector de R1) ror (integer de N)*
15	MULT	Cap	Multiplica R1 i R2, i deixa el resultat en R1.
15	DIV	Cap	Fa la divisió entera de R1 entre R2 i deixa el resultat de la divisió entera en R1.

11	JMPNR1 EXT	Posició de memòria	Salta a la posició de memòria indicada si R1 és negatiu. .
11	JMPZR1 EXT	Posició de memòria	Salta a la posició de memòria indicada si R1 és 0..
12	R3ADD	Cap	Suma R1 i R2, i deixa el resultat en R3 (el registre R3 s'ha de crear prèviament en la UP). .

Per fer el *casting* o conversió de formats disposem de les funcions següents:

Funció	Argument	Retorna
conv_std_logic_vector	integer	std_logic_vector
conv_integer	std_logic_vector	integer
to_bit	std_logic	bit
to_stdlogic	bit	std_logic

3. Executeu el codi següent, i presenteu el resultat de la simulació tenint en compte que es vol monitoritzar el resultat de tots els registres; en format decimal signat per a R1 i R2 i en format decimal sense signe per a la resta.

```
DEPTH = 255;
WIDTH = 12;
ADDRESS_RADIX = DEC;
DATA_RADIX = HEX;
-- Recordeu que heu de posicionar el codi a partir de l'adreça adient.
CONTENT
BEGIN
0       :     000;
1       :     110;     -- LDR2 16
2       :     011;     -- LDR1 17
3       :     700;     -- SUBS
4       :     011;     -- LDR1 17
5       :     400;     -- CMP
6       :     501;     -- BRE 1
7       :     200;     -- ADD
8       :     601;     -- JMP 1
[9..15]:     000;
16      :     003;      % A partir d'aqui dades %
17      :     00B;
18      :     002;
19      :     005;
[20..255]:  000;
END;
```

4. Hi ha alguna manera d'optimitzar (fer-lo amb menys microinstruccions) el cicle de cerca d'instrucció? Raoneu breument la resposta.

5. Hi ha alguna manera d'optimitzar (fer-lo amb menys microinstruccions) el cicle d'execució de la instrucció STR1? Raoneu breument la resposta.

   Indicació: per respondre les preguntes 4 i 5 es recomana estudiar el comportament de la memòria RAM.

6. Quins canvis significatius comportaria el fet de fer servir una unitat de procés de 3 busos (vegeu la Fig. 16.3) en lloc d'una de 2 busos com a la pràctica.
   Enumereu tots el canvis i feu una previsió del nombre de cicles necessaris per tal d'implementar:
   — el *fetch*
   — la instrucció ADD
   — la instrucció LDR1

7. Suposem que s'està executant la microinstrucció de la posició 0 de la ROM. Quina seria l'adreça de la microinstrucció següent a executar? Justifiqueu breument la resposta. I si estiguéssim executant la posició 6? I la posició 24? I la posició 19?

8. Suposem que el processador està executant un programa emmagatzemat a la RAM. Es pot observar que habitualment, durant la realització dels cicles de cerca, canvia el valor generat pel mòdul IRdecoder. Quina n'és la causa?
   Si en un cicle de cerca el valor generat per l'IRdecoder no canvia, què se'n pot deduir?

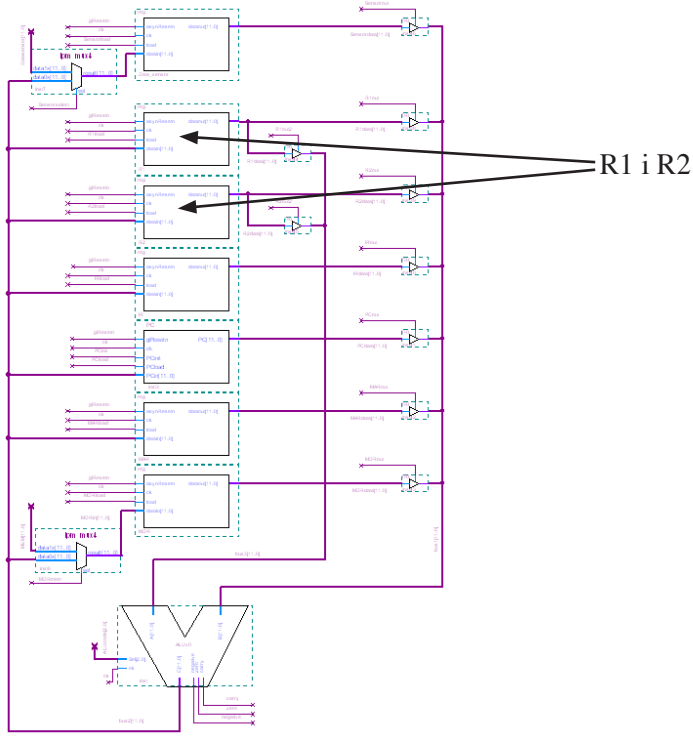


Fig. 16.3 Unitat de procés amb 3 busos

9. Quines són les diverses adreces que pot carregar el registre microPC? D'on provenen? Com van sincronitzades la càrrega del microPC, la lectura de la ROM i la càrrega dels registres de la UP?

10. Reduïu el nombre de bits de control dels tristates de sortida del registres, que accedeixen al BUS1 decodificant els seus bits de control.

    Consideracions: fent aquesta descodificació es pot reduir de 6 senyals de control dels REGout a només 3 senyals. Reduïu el nombre de bits total de la UC així com la mida de la paraula de la ROM.

11. Feu que el processador, en comptes de poder implementar només 16 instruccions, pugui implementar fins a 128 instruccions.

12. Convertiu el processador de 12 bits en un nou processador de 32 bits. Heu de retocar tota l'arquitectura del processador.

16.4. Qüestionari escrit – Capítols 14 - 15

1. Digueu respecte a la FPGA què hi ha a la placa:
   — Quin model s'ha utilitzat?
   — Quants elements lògics (en anglès, *Logic Elements*, LE) incorpora?
   — Quants LE ha ocupat el vostre microprocessador? (Indicació: aquest valor surt quan es fa una síntesi)
   — Quants pins té l'FPGA?
   — Quin tant per cent d'aquests pins es fan servir?

   Si alguna dada no la trobeu mireu la pàgina web d'Altera.

2. Enumereu quines memòries externes té la placa. Fem servir aquestes memòries per a la implementació de LittleProc? Raoneu breument la resposta.

3. Com ho heu fet per tal de visualitzar el rellotge del sistema amb la placa de prototipatge?

4. Per què és necessari l'arxiu *pins.tcl*? Es pot simular el muntatge sense aquest arxiu? I es pot baixar a placa? Raoneu molt breument les respostes.

5. Expliqueu la diferència entre els senyals pulse, clk i ck.

6. Expliqueu en què consisteix l'etapa de programació quan es treballa amb FPGA. Hi ha cap fitxer que tingui una importància especial en aquesta etapa?

7. Donat aquest programa i suposant que el PC quan és inicialitzat ho fa amb el valor 1:

```
#0    000;
#1    LDR2 8
#2    LDR1 9
#3    SUBS
#4    STR1 19
#5    END
…
#8    014;
#9    010;
```

a) Quant de temps triga a executar-se aquest programa? Raoneu i mostreu els càlculs pertinents.
b) Si els valors de les posicions de memòria 8 i 9 canvien, el temps d'execució també canvia? Raoneu la resposta.

16.5. Preguntes per a qüestionaris orals – Capítols 1 - 6

Preguntes conceptuals

1. Què és un circuit combinacional?
2. I un circuit seqüencial?
3. Funciona el circuit X per flanc o per nivell?
4. Quines diferències hi ha entre una balda i un flip-flop?
5. Es pot fer un comptador amb baldes? I amb flip-flops?

Preguntes relatives a una cel·la desenvolupada mitjançant captura d'esquemes

1. Explica qualsevol aspecte del comportament de la cel·la.
2. Mostra una simulació realitzada i explica-la.
3. Quantes portes de tipus X hi ha en aquesta cel·la?
4. Quines entrades i quines sortides té aquesta cel·la?
5. L'entrada X és activa a alta o a baixa? És síncrona o asíncrona?
6. Aquesta cel·la és combinacional o seqüencial?
7. Quin tipus de flip-flops s'utilitzen en aquesta cel·la?
8. Per a què serveixen els *buffers* presents a la cel·la ctrlSeq?
9. Com es podria realitzar una inicialització asíncrona del registre PC a un valor determinat?
10. Què signifiquen els cercles que apareixen en algunes entrades de certes portes?

Preguntes sobre VHDL

1. Què és una entitat?
2. Per a què serveix la sentència *architecture*?
3. Què és un *package*? Digues algun *package* que hagis utilitzat en el teu disseny.
4. Quines diferències hi ha entre un senyal i una variable?
5. Es pot usar una variable per intercomunicar dos processos?
6. Què és la llista de sensibilitat? Quins elements la formen en el cas d'un mòdul combinacional? Com s'expressa una assignació concorrent condicional en VHDL?
7. Per a què serveix la clàusula *when others* en una sentència *case*? Quan és obligatori posar-la?
8. Depèn VHDL de l'ús de majúscules i minúscules?

9. Com es descriu una màquina d'estats de Moore mitjançant tres processos?
10. Explica els tipus *bit*, *bitvector*, *stdlogic* i *stdlogicvector*. Diferències entre ells.
11. Per a què serveixen les sentències *attribute* en les descripcions VHDL?

Preguntes sobre cel·les desenvolupades mitjançant VHDL

1. Explica el codi de la cel·la.
2. És una determinada cel·la combinacional o seqüencial?
3. Mostra una simulació realitzada i explica-la.
4. Explica qualsevol operació de l'ALU
5. Explica la descripció de l'ALU mitjançant dos processos. Per a què serveix cada procés?
6. Explica la llista de sensibilitat de qualsevol procés.

16.6. Preguntes per a qüestionaris orals – Capítols 7 - 13

1. Quants registres té la UP? I la UC?
2. Donat el contingut d'una microinstrucció de la ROM, com se'n pot saber el tipus?
3. S'executa una microinstrucció de tipus EXE que està en una posició de la ROM. Quina és l'adreça de la següent microinstrucció a executar?
4. S'executa una microinstrucció de tipus IF que està en una posició de la ROM. Quina és l'adreça de la següent microinstrucció a executar?
5. S'executa una microinstrucció de tipus GOTO que està en una posició de la ROM. Quina és l'adreça de la següent microinstrucció a executar?
6. Quines condicions es poden usar en les microinstruccions de tipus IF a la unitat de control? Per què?
7. Quins elements hi ha en el mòdul seqüenciador? Per a què serveix cadascun?
8. En quin element es desa l'estat de la unitat de control?
9. Suposem que s'executa una microinstrucció i com a conseqüència d'això el senyal W2 adopta el valor 1. Quin és el tipus de la microinstrucció que s'acaba d'executar?
10. Suposem que eliminéssim les portes AND de la unitat de control. Hi hauria cap problema?
11. Suposem que es volgués poder treballar amb un senyal de condició addicional. Quins elements de la ROM s'haurien de modificar? En què consistirien aquestes modificacions?
12. Per a què serveixen els senyals W3 i W1? I el senyal W2?
13. En quins casos una microinstrucció de tipus IF pot contenir senyals de control de la UP?
14. En quins casos una microinstrucció EXE pot contenir una adreça de memòria?
15. És el mòdul ctrlSeq un circuit combinacional o seqüencial? Per què? És el seqüenciador un circuit combinacional o seqüencial? Per què?
16. Què passa en el cicle d'inicialització del processador?
17. Què passa en el cicle de cerca de les instruccions? Descriu-lo microinstrucció a microinstrucció.
18. Què passa en el cicle d'execució de la instrucció X? Descriu-lo microinstrucció a microinstrucció.
19. Què diferencia una microinstrucció de tipus GOTO d'una BRA?
20. Què diferencia una microinstrucció de tipus IF d'una de tipus GOTO?
21. En una microinstrucció de tipus BRA, com es determina l'adreça de la següent microinstrucció a executar.

22. LittleProc compta amb un bon nombre de *buffers* (LCELL). Per a què els utilitzem?
23. Quina finalitat tenen les sentències *attribute* en el mòdul *incrementador*?
24. Quants busos té la unitat de procés de LittleProc?
25. És possible realitzar en un sol cicle de rellotge ACC ← mask(IR)?
26. Quants camins directes hi ha entre la UP i la RAM? D'on a on van?
27. D'on pot rebre les dades el MDR? I l'IR? I l'ACC?
28. Per a què serveix el registre PC? I l'IR? I el MAR? I el MDR? I l'ACC?
29. Quantes instruccions té LittleProc?
30. En una simulació, quin és el format adequat per veure un senyal que constitueix una adreça de la ROM? I per veure el contingut de l'IR? I per veure la codificació de l'operació de l'ALU?
31. Per a què serveix l'IRDecoder? Quin mòdul li envia les seves dades d'entrada?

16.7. Preguntes per a qüestionaris orals – Capítols 14 - 15

1. Enumera les modificacions que s'han hagut de fer a LittleProc per procedir a la seva implementació en placa.
2. Quins són els components principals de la placa UP3/DE2?
3. Quins elements concrets de la placa han estat utilitzats directament per a la implementació de LittleProc?
4. Com s'ha implementat la ROM i la RAM de LittleProc?
5. Què caldria fer si es volgués utilitzar un polsador diferent per aplicar el senyal *startn*?
6. En què consisteix l'etapa de programació d'una FPGA?
7. Què són els fitxers amb l'extensió .sof?
8. Quins passos cal seguir per vincular uns pins de l'FPGA als terminals del circuit i implementar el circuit en l'FPGA?
9. Quants dígits hexadecimals s'usen per visualitzar el contingut dels registres en el *display*? És el mateix en tots els casos? Quina tensió d'alimentació requereix la placa UP3/DE2?
10. Què és una FPGA? Quins són els principals elements d'una FPGA?
11. Quins són els dos principals fabricadors de FPGA? De quin fabricant és l'FPGA que ha estat utilitzada per a la implementació de LittleProc?
12. A quina família pertany l'FPGA que ha estat usada?
13. En quin moment del procés de disseny s'especifica el model de FPGA que s'usarà?
14. Quin problema hi hauria si uséssim el rellotge de la placa com a rellotge de LittleProc (el rellotge que regeix la UC i la UP)?
15. On es connecten els extrems del cable que comunica la placa UP3/DE2 i l'ordinador?
16. Explica el cicle d'execució de la instrucció addicional que ha desenvolupat el vostre grup per al qüestionari escrit.
17. Si es volgués afegir una instrucció addicional a LittleProc, quins canvis caldria fer-hi i en quins elements?

17. Referències i bibliografia

17.1. Referències relacionades amb els capítols 0 - 5

Lloris, A.; Prieto, A. 1996. Diseño Lógico. Ed. McGraw-Hill Interamericana.
Gajski ; D. D. 1197. Principles of Digital Design. Ed. Prentice-Hall.
Brown, R. B.; Lomax, R. J.; Carichner, G.; Drake, A. J. «A Microprocessor Design Project in an Introductory VLSI Course». IEEE Transactions on Education, vol. 43, núm. 3, paàg. 353-61.
Deschamps, J. P. 2002. Síntesis de Circuitos Digitales, cap. 12, Ed. Thomson-Paraninfo.
Heuring, V. P.; Jordan, H. F. 2004. Computer Systems Design and Architecture, cap. 4 i 5. Ed. Pearson Prentice Hall.
Pàgina principal de Moodle: https://moodle.org. Setembre de 2015.
Còpia del servidor Moodle emprat al curs de Disseny de Sistemes Digitals: http://ume015.uab.es/dsdmoodleRITA. Setembre de 2015.
Cunningham, W.; Leuf, B. 2001. The Wiki Way, Addison-Wesley.
Introduction to the Quartus II Software, Altera Corporation.
Pàgina web d'Altera: http://www.altera.com. Setembre de 2015.

17.2. Referències VHDL

Uyemura, J. P. Introducción al Diseño de Sistemas Digitales. Un enfoque integral. International Thomson Editores.
Deschamps, J. P. Síntesis de Circuitos Digitales: un enfoque algorítmico. Thomson Editores.
Terés, L.; Torroja, Y.; Olcoz, S.; Villar, E. 1998. VHDL Lenguaje estándar de diseño electrónico. Ed. McGrawHill.
Ashenden, P. J. The Designer's Guide to VHDL (Systems on Silicon). Morgan Kaufmann Publishers.
IEEE Standard VHDL Language Reference Manual IEEE Std 1076-2000 Volume, Issue, 2000 Page(s):i – 290

17.3. Referències relacionades amb la implementació a placa

UP3 Education Kit. Reference Manual, Cyclone Edition. Altera Corporation i System Level Solutions. Agost de 2004.
UP3-1C12 Education Kit. Reference Manual, Cyclone Edition. Altera Corporation i System Level Solutions. Març de 2005.

Introduction to DE2. Altera Corporation. 2006. Es pot accedir a aquest document (que a més d'informació tècnica també conté informació comercial) en l'adreça ftp://ftp.altera.com/up/pub/Webdocs/DE2_Introduction.pdf. Setembre de 2015.

DE2 Development and Education Board. User Manual. Altera Corporation. 2006. Es pot accedir a aquest manual en l'adreça ftp://ftp.altera.com/up/pub/Webdocs/DE2_UserManual.pdf. Setembre de 2015.

Tota la informació complementària, la fe d'errades i el programari necessari per a les pràctiques els podeu trobar a: http://ume015.uab.es/llibre. Setembre de 2015.

17.4. Publicacions en congressos i revistes

Portero, A.; Saiz, J.; Aragonés, R. 2006. «Transforming Spanish Student Attitude in the face of Engineering learning», Proceedings of the 10th IACEE World Conferer-ence on Continuing Engineering Education, abril, Viena (Àustria). http://www.ai.tuwien.ac.at/wccee2006/proceedings/papers/390.pdf. Setembre de 2015.

Portero, A.; Saiz, J.; Aragonés, R. 2006. «Convergencia hacia el EEES en Sistemas Digitales de Ingeniería Informática», Congrés CIDUI, UPC, Barcelona.

Aragonés, R.; Saiz, J.; Portero, A. 2008. «Experiencia de innovación docente siguiendo las directrices del Espacio Europeo de Educación Superior en la enseñanza del diseño digital». Revista Latinomaricana de Tecnología Educativa. http://dialnet.unirioja.es/servlet/fichero_articulo?articulo=2229199&orden=89170. Setembre de 2015.

Saiz J.; Portero, A.; Aragonés, R. 2009. «Máquinas Algorítmicas: una metodología para su aprendizaje práctico a través del LittleProc», IEEE-Revista Iberoamericana de Tecnologías del Aprendizaje, maig. http://rita.det.uvigo.es/200905/uploads/IEEE-RITA.2009.V4.N2.A1.pdf. Setembre de 2015.

Llista de figures i taules

Taula 0.1 Planificació de sessions 12
Fig. 1.1 Obertura de Quartus II Web Edition 15
Fig. 1.2 Menú vertical d'ordres d'obertura i tancament de fitxers i projectes 16
Fig. 1.3 Creació d'un nou projecte: finestra inicial informativa 18
Fig. 1.4 Creació d'un nou projecte: introducció del directori, nom i cel·la principal 18
Fig. 1.5 Creació d'un nou projecte: addició de fitxers al projecte 19
Fig. 1.6 Creació d'un nou projecte: especificació del dispositiu EP1C12Q240C8 .. 20
Fig. 1.7 Creació d'un nou projecte: especificació del dispositiu EP2C35F672C6 ... 20
Fig. 1.8 Creació d'un nou projecte: especificació d'eines CAD addicionals 21
Fig. 1.9 Creació d'un nou projecte: finestra resum del projecte 21
Taula 2.1 Taula de veritat de ctrlSeq 23
Fig. 2.1 Esquema lògic de ctrlSeq 23
Fig. 2.2 Formulari per a la creació d'un nou esquema 24
Fig. 2.3. Finestra de l'editor d'esquemes 24
Fig. 2.4 Addició d'un nou símbol 25
Fig. 2.5 Posicionament de cel·les 27
Fig. 2.6 Posicionament de pins 27
Fig. 2.7 Addició de línies de connexió 28
Fig. 2.8 Símbol generat automàticament 29
Fig. 2.9 Modificació de l'ordre dels pins en el símbol 30
Fig. 2.10 Símbol definitiu de ctrlSeq 30
Fig. 3.1 Botó de compilació 31
Fig. 3.2 Finestra informativa del progrés de la compilació 31
Fig. 3.3 Execució separada de les diverses etapes de la compilació 32
Fig. 3.4 Exemple d'avisos produïts durant la compilació 32
Fig. 3.5 Exemple d'errors produïts durant la compilació 32
Fig. 4.1 Inicialització de les opcions relatives a les formes d'ona 33
Fig. 4.2 Formulari amb les opcions de simulació (s'ha escollit fer una simulació funcional) 34
Fig. 4.3 Finestra d'edició de formes d'ona 35
Fig. 4.4 Addició d'entrades a l'editor de formes d'ona (I) 36
Fig. 4.5 Addició d'entrades a l'editor de formes d'ona (II) 36
Fig. 4.6 Addició d'entrades a l'editor de formes d'ona (III) 37
Fig. 4.7 Addició d'entrades a l'editor de formes d'ona (IV) 37
Fig. 4.8 Editor de formes d'ona amb les entrades inicialitzades al valor lògic zero 37
Fig. 4.9 Botons d'ordres d'edició d'ones 38
Fig. 4.10 Generació de senyals periòdics 39
Fig. 4.11 Estímuls d'entrada finalitzats 39
Fig. 4.12 Botó de simulació 39
Fig. 4.13 Finestra amb els resultats de la simulació funcional 40

Fig. 4.14 Formulari amb les opcions de simulació (s'ha escollit fer una simulació amb retards) ... 41
Fig. 4.15 Finestra d'edició de formes d'ona ... 41
Fig. 4.16 Especificació de la durada de la simulació ... 42
Fig. 4.17 Especificació de la grandària de la graella de l'editor de formes d'ona ... 42
Fig. 4.18 Addició d'entrades a l'editor de formes d'ona (I) ... 43
Fig. 4.19 Addició d'entrades a l'editor de formes d'ona (II) ... 43
Fig. 4.20 Addició d'entrades a l'editor de formes d'ona (III) ... 43
Fig. 4.21 Addició d'entrades a l'editor de formes d'ona (IV) ... 44
Fig. 4.22 Creació d'un grup de senyals ... 45
Fig. 4.23 Botó de l'ordre de definició d'un comptador ... 45
Fig. 4.24. Formulari associat a la definició d'un comptador ... 45
Fig. 4.25 Entrades agrupades amb comptador associat ... 46
Fig. 4.26 Botons per a l'especificació de valors constants a les entrades ... 47
Fig. 4.27 Botó per a la generació d'un senyal periòdic ... 47
Fig. 4.28 Formulari associat a la generació d'un senyal periòdic ... 47
Fig. 4.29 Especificació d'una forma d'ona de manera lliure ... 47
Fig. 4.30 Botons per fixar el valor d'un tram d'ona ... 47
Fig. 4.31 Especificació del fitxer d'estímuls ... 48
Fig. 4.32 Botó de simulació ... 48
Fig. 4.33 Finestra amb els resultats de la simulació amb retards ... 49
Fig. 4.34 Senyals d'entrada per a la segona simulació addicional de ctrlSeq ... 49
Fig. 5.1 Balda ... 52
Fig. 5.2 Forma d'ones d'una balda ... 52
Fig. 5.3 Registre de 4 baldes ... 53
Fig. 5.4 Símbol del registre de 4 baldes ... 53
Fig. 5.5 *Flip-flop* D amb preset i reset asíncrons ... 54
Fig. 5.6 Formes d'ona d'un *flip-flop* ... 54
Fig. 5.7 Exemple de connexió per nom ... 55
Fig. 5.8 Formulari per introduir el nom d'un node elèctric ... 55
Fig. 5.9 Assignació de nom a una línia de maneres correcta i incorrecta ... 55
Fig. 5.10 Registre de *flip-flops* amb especificació de busos ... 56
Fig. 5.11 Registre de *flip-flops* de 4 bits expressat de manera compacta ... 57
Fig. 5.12 Cel·la DFF ... 58
Fig. 5.13 Cel·la DFFE ... 58
Fig. 6.1 Creació del fitxer VHDL ... 60
Fig. 6.2 Entrada escrita en VHDL de l'exemple adder.vhd ... 60
Fig. 6.3 Diagrama d'un bloc lògic ... 61
Fig. 6.4 Ús d'un compilador VHDL ... 62
Fig. 6.5 Diagrama d'un MUX de 2 entrades ... 64
Taula 6.1 Operadors ... 68
Fig. 6.6 adder.bdf ... 71
Fig. 6.7 Xor3 a partir de 2 xor2 ... 71
Fig. 6.8 Disposició dels diferents leds en un 7 segments ... 74
Fig. 6.9 Formes d'ona d'un conversor bcd -7 segments ... 75
Fig. 6.10 Diferents representacions d'una màquina d'estats. La descripció de VHDL es fa amb un sol procés ... 78
Fig. 6.11 Diferents representacions d'una màquina d'estats. La descripció de VHDL es fa amb dos processos ... 79
Fig. 6.12 Máquina de Moore. El vector de sortida és funció de l'estat de la màquina: Y = f(STATE) ... 81

Fig. 6.13 Màquina de Moore amb 2 i 3 processos 81
Fig. 6.14 Exemple de màquina de Moore (diagrama d'estats i codi VHDL) 81
Fig. 6.15 Formes d'ona de la màquina de Moore 83
Fig. 6.16 Màquina de Mealy amb 2 i 3 processos 83
Fig. 6.17 Exemple de màquina de Mealy (diagrama d'estats i codi VHDL)..... 84
Fig. 6.18 Formes d'ona de la màquina de Mealy 85
Fig. 6.19 Símbol del mòdul inc 85
Fig. 6.20 Símbol del mòdul reg 86
Fig. 6.21 Símbol mòdul de la unitat aritmeticològica 87
Fig. 6.22 Possible diagrama de la unitat aritmeticològica 87
Fig. 6.23 Rotació a la dreta incloent-hi el carry 88
Fig. 6.24 Desplaçament a la dreta incloent-hi el carry 89
Fig. 6.25 Taula de conversió i símbol del bloc IRdecoder 90
Fig. 7.1 Pinout extern del processador 91
Fig. 7.2 Estructura interna del processador 92
Fig. 7.3 Estructura interna de la unitat de procés 93
Fig. 7.4 Buffer tristate per a la sortida del registre R2 94
Fig. 7.5 Multiplexor a l'entrada del MDR 94
Fig. 7.6 Exemple del registre R2 95
Fig. 7.7 Exemple del registre PC 96
Fig. 7.8 Multiplexació de les dades d'entrada del MDR 97
Fig. 7.9 Procés de creació d'un multiplexor de N bits d'entrada i M canals 97
Fig. 7.10 Part final del procés de creació del multiplexor del MDR 98
Fig. 7.11 Multiplexor del MDR 98
Fig. 7.12 Unitat aritmeticològica 99
Fig. 7.13 Esquema intern de la unitat de procés 100
Fig. 7.14 Selecció dels nodes interns 102
Fig. 7.15 Implementació de les quatre primeres línies del guió de la simulació ... 104
Fig. 7.16 Preparació de la simulació: senyals ja ordenats 105
Fig. 8.1 Esquema intern de la unitat de control 108
Fig. 8.2 Esquema intern del seqüenciador 109
Fig. 8.3 Multiplexor d'entrades del seqüenciador 110
Fig. 8.4 microPC (imatge girada: entrades a la dreta) 110
Fig. 8.5 L'incrementador 111
Fig. 8.6 IRdecoder 112
Fig. 8.7 Multiplexor de condicions d'entrada 112
Taula 8.1 Taula de control del seqüenciador (g és una entrada de ctrlSeq indicadora de l'estat d'una determinada condició) 113
Fig. 8.8 Símbol de ctrlSeq 114
Fig. 8.9 Wizard de creació d'una ROM 115
Fig. 8.10 Símbol de la ROM 117
Fig. 8.11 Cronograma de sortida de la ROM 117
Fig. 8.12 Esquemàtic de la UC 118
Fig. 8.13 Especificació dels àlies dels senyals de sortida 119
Taula 9.1 *Instruction set* del processador (inici) 123
Taula 9.1 *Instruction set* del processador (continuació) 124
Taula 10.1 Taula de correspondència de la ROM (inici) 127
Taula 10.1 Taula de correspondència de la ROM (continuació) 128
Taula 10.2 Taula de descodificació de les instruccions 129
Fig. 10.1 Fitxer de la ROM 130
Fig. 11.1 Memòria RAM d'instruccions/dades 131

Fig. 11.2 Fitxer de la RAM: especificació de la capçalera 132
Fig. 11.3 Fitxer de la RAM: especificació d'un programa d'exemple 133
Fig. 11.4 Cronograma amb cicles de lectura de la RAM 133
Fig. 11.5 Cronograma amb un cicle d'escriptura a la RAM 134
Fig. 12.1 Esquema global del processador 135
Taula 12.1 Mapeig de senyals de control 136
Fig. 13.1 Diagrama de la unitat de procés 139
Fig. 13.2 Unitat de procés d'un sol bus 140
Taula 13.1 Taula de la ROM (inici) 141
Taula 13.1 Taula de la ROM (continuació) 142
Fig. 14.1 Esquema bàsic d'una placa amb una FPGA 143
Fig. 14.2 Placa de prototipatge UP3 Education Kit 144
Fig. 14.3 Visió de l'esquema de LittleProc 145
Fig. 14.4 Modificacions de LittleProc per implementar-lo a placa 146
Fig. 14.5 Modificació del rellotge del sistema (clock.vhd) 146
Fig. 14.6 Mòdul controlador de la pantalla LCD (lcd.vhd) 147
Fig. 14.7 Display de 16x2 caràcters 147
Fig. 14.8 Esquemàtic de la UP original 148
Fig. 14.9 Esquemàtic de la UP modificada 148
Fig. 14.10 UC modificada: pin de sortida afegit i nova connexió a l'entrada opcode 149
Fig. 14.11 Associació de pins 150
Fig. 14.12 Programador 150
Fig. 14.13 Finestra de programació. La interfície de programació és ByteBlasterII 151
Fig. 14.14 ByteBlasterII mode JTAG 151
Fig. 14.15 Finestra per connectar port paral·lel 151
Fig. 14.16 Finestra per programar la placa: càrrega del fitxer .sof 152
Fig. 15.1 Esquema bàsic d'una placa amb una FPGA 153
Fig. 15.2 Fotografia de la placa Altera DE2 (extreta del manual de la placa) 155
Fig. 15.3 LittleProc: circuit inicial 156
Fig. 15.4 LittleProc: circuit modificat 157
Fig. 15.5 Modificació del rellotge del sistema: addició del mòdul *clock* 158
Fig. 15.6 Unitat de procés: símbol i connectivitat inicials 159
Fig. 15.7 Unitat de procés: símbol i connectivitat modificats 159
Fig. 15.8 Unitat de control: símbol i connectivitat modificats 160
Fig. 15.9 LCD: presentació del contingut dels registres 160
Fig. 15.10 Mòdul controlador de l'LCD 161
Fig. 15.11 Associació de pins 162
Fig. 15.12 Botó per a la programació de la placa 163
Fig. 15.13 Finestra de programació. La interfície de programació és USB-Blaster 163
Fig. 15.14 Especificació de la interfície de programació USB-Blaster 163
Fig. 15.15 Càrrega del fitxer LittleProc.sof 164
Fig. 16.1 Jerarquia d'un circuit 165
Fig. 16.2 Significat dels cercles en els símbols 166
Fig. 16.3 Unitat de procés amb 3 busos 171